

# Tractable Combinatorial Auctions Via Graph Matching

**Frank Guerin**

FGUERIN@CSD.ABDN.AC.UK

*Department of Computing Science, University of Aberdeen,  
Aberdeen, AB24 3UE, Scotland*

## Abstract

Combinatorial auctions play a key role in distributed AI as a mechanism for efficiently allocating resources or tasks, both for cooperative and competitive scenarios. The complexity of the general problem has lead to interest in finding efficient algorithms for useful instances. Tennenholtz introduced an approach which finds polynomial solutions for certain classes of combinatorial auctions by a reduction to graph matching problems. This paper looks at the possibilities and limits of this graph-matching approach. We extend the set of known tractable combinatorial auctions identified by Tennenholtz to accommodate subadditive symmetric bids over bundles of unlimited size, certain restricted cases of asymmetric discounts over bundles, and certain restricted cases of superadditive bids. We provide results on the ultimate limits of the approach, identifying classes of auctions which cannot be reduced to 2D graph matching, and classes which are NP-equivalent. We provide complexity results for the solution of the identified classes of tractable auctions, using the current best available graph matching algorithms.

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Preliminaries</b>	<b>5</b>
2.1	Combinatorial Auctions . . . . .	5
2.2	2D-Graph Matching Algorithms for Auctions . . . . .	6
<b>3</b>	<b>Graphical Structures for Various Bids</b>	<b>9</b>
3.1	Bids for Multiple Objects with a Quantity Constraint . . . . .	10
3.2	Bids for Subadditive Pairs . . . . .	11
3.3	Almost Additive Bids . . . . .	13
3.4	Bids for Subadditive Triples . . . . .	14
3.4.1	The Case Where $d_3 \geq 2d_2$ . . . . .	14
3.4.2	The Case Where $d_2 \leq d_3 < 2d_2$ . . . . .	15
3.4.3	The Case Where $d_3 < d_2$ . . . . .	22
3.5	Bids for Subadditive Quadruples . . . . .	23
3.5.1	The Case Where $d_4 \geq 2d_3 - d_2$ and $d_3 \geq 2d_2$ . . . . .	23
3.5.2	The Case Where $d_3 \geq d_2$ and $d_4 \geq d_3 + d_2$ . . . . .	24
3.5.3	The Case Where $d_4 \geq d_3 > 2d_2$ . . . . .	24
3.6	Bids for Subadditive $n$ -Tuples . . . . .	24
3.7	Asymmetric Subadditive Bids . . . . .	26
3.8	Superadditive Bids . . . . .	27
3.8.1	Note on Tennenholtz's Superadditive Auction . . . . .	33
<b>4</b>	<b>The Space of Possible Bids</b>	<b>34</b>
4.1	Pair Space . . . . .	34
4.2	Triple Space . . . . .	36
4.2.1	Right Face of Unit Cube . . . . .	37
4.2.2	Top Face of Unit Cube . . . . .	37
4.2.3	Back Face of Unit Cube (Excluding Unknown Boundaries) . . . . .	38
4.2.4	Unknown Boundaries and Regions . . . . .	39
4.3	Quadruple Space (and beyond) . . . . .	40
4.4	Discussion of Limitations . . . . .	42
<b>5</b>	<b>Complexity Results</b>	<b>44</b>
5.1	Optimal Winner Determination in Auctions with Bipartite Structures . . .	44
5.2	Optimal Winner Determination in Auctions with Nonbipartite Structures .	44
5.3	Efficient Handling of Quantity Constrained and Multi-Unit Bids . . . . .	45
<b>6</b>	<b>Related Work</b>	<b>47</b>
<b>7</b>	<b>Conclusion</b>	<b>50</b>
	<b>References</b>	<b>51</b>

## 1. Introduction

Auctions are a ubiquitous market mechanism, allowing efficient matching of buyers and sellers. The rise of the Internet introduces new opportunities for their use. In many traditional markets, practical constraints have limited the extent to which auctions could be deployed. For example without computer support it is not feasible to rapidly handle the large volumes of information required for fine grained bidding which can express valuations over combinations of objects, nor is it feasible to cope with dynamic changes in price, nor to calculate optimum allocations and pricing. In the absence of computer support, more rigid pricing mechanisms have to be employed, leading to inefficient allocations and ultimately to wastage of resources. The Internet, and electronic technologies more generally, now have the potential to facilitate fine grained and efficient dynamic pricing and allocation. Any improvement in the efficiency of such a ubiquitous mechanism has the potential to bring great benefits to society; this explains the considerable amount of research effort directed to this end.

The auction is a very general idea, applying not only to the sale of goods, but also to the allocation of resources (for example the allocation of computing resource in a grid computing scenario) and the allocation of tasks (agents offer to take on combinations of tasks and to complete them to some stated specification). Finding efficient allocations is equally important in cooperative and competitive scenarios. All of this underscores the key role of auctions, and in particular combinatorial auctions, in distributed AI.

The desire to express combinatorial bids arises naturally in many scenarios. For example, two consecutive advertising slots may be more valuable than two non-consecutive slots; the consecutive slots would allow a different class of advertisement to be run. A bidder may be willing to pay more for the two consecutive slots than the sum of what he would be willing to pay for each one independently. This increase in value occurs when the combined objects complement one another; this is called “superadditive”. On the other hand substitutable goods tend to have a decreasing value per unit as bundles of larger and larger sizes are considered. In this case the value of combined bundles is less than the sum of the individual objects’ values; this is called “subadditive”. Sometimes subadditive bids will be expressed by sending the auctioneer a function describing a downward curve (price per unit as a function of number of units). More complex interactions are also possible, for example buying more and more chunks of radio spectrum may lead to diminishing returns, but to be allocated the whole lot might be extremely valuable as it excludes competitors. Combinatorial bidding allows bidders to precisely express their valuations; however, the downside is the computational complexity, both of the bidding languages and the allocation problem. A fully expressive bidding language will allow an agent to express a valuation for every possible combination of a set of objects. The size of such bids obviously grows exponentially with the size of the set. Likewise the allocation problem quickly becomes computationally difficult. The allocation problem is commonly known as the CAP (combinatorial auction problem); it is in general NP-hard, and has been shown to be NP-hard even for very restricted cases such as bundles of size  $\leq 3$  (Rothkopf, Pekec, and Harstad (1998) show that a special case of such an auction is an instance of maximal 3-set packing). These severe computational difficulties look set to continue to provide researchers in this area with a steady stream of research challenges for some time.

Nisan (2000) introduced the following three-way classification for current work on allocation algorithms for combinatorial auctions:

1. Polynomial instances of the CAP: finding computationally tractable special cases via restricted classes of bids, for example Rothkopf et al. (1998), Sandholm and Suri (2001), Conitzer, Derryberry, and Sandholm (2004).
2. Clever search algorithms for the full CAP, for example Sandholm, Suri, Gilpin, and Levine (2001). A common tactic is to exploit the fact that bidding tends to be sparse in practice.
3. Approximation algorithms for the full CAP: heuristic-based algorithms giving almost optimal allocations in polynomial time, for example Dang and Jennings (2003), Sakurai, Yokoo, and Kamei (2000).

This paper belongs to the first category. Tennenholtz (2000, 2002) introduced an approach which finds polynomial solutions for certain classes of combinatorial auctions by a reduction to graph matching problems. Using this approach he has identified new classes of tractable combinatorial auctions, and shown how they can be solved efficiently using  $b$ -matching techniques. We explore this approach further to find the possibilities and limits; we find new tractable instances and new ways to solve them using simpler weighted graph matching techniques in place of  $b$ -matching. This paper makes four contributions:

1. It extends the set of known “tractable combinatorial auctions” identified by Tennenholtz (2002); in particular, to accommodate bundles of unlimited size, certain restricted cases of asymmetric discounts over sub-bundles, and certain restricted cases of superadditive bids.
2. It provides results on the limits of these graph matching approaches to representing auctions; i.e. it locates the boundary which divides the classes of auctions which can and cannot be solved with the graph matching techniques under consideration.
3. In addition to extending the set of known tractable auctions, it also closes the gap “from the other side”, expanding the set of known NP-equivalent auctions. In between the tractable and NP-equivalent, the paper outlines the “unknown” region, which we have proven cannot be solved by our matching techniques, and for which we do not know any tractable solution method, but which have not been proven to be NP-equivalent.
4. It provides complexity results for solving the identified classes, by using the current most efficient graph matching techniques available.

We can summarise the results by saying that graph matching techniques are an excellent way to handle many subadditive combinatorial auctions, but are quite limited for superadditive auctions. Most of these superadditive cases are shown to be NP-hard, and graph matching seems to take us quite close to the boundary of known NP-hardness, with the unknown region in between being relatively small. It is hoped that these results will provide researchers in the area with valuable knowledge of the types of auction that graph

matching approach are good for, and how proposed auctions might be constrained so as to be efficiently solvable by matching techniques. The results may also contribute to general combinatorial auction solvers (such as described by Sandholm (2002)), where general bids are accepted, but the solver recognises special cases that can be handled by known polynomial algorithms.

Throughout the paper we will aim to make topics accessible to the general AI reader who may have limited familiarity with graph theory. We will not follow the graph theory conventions of  $m$  for number of edges and  $n$  for number of vertices, preferring  $E$  and  $V$ . To minimise clutter we will omit the vertical bars to denote the size of a set where it is obvious that size is being referred to; e.g. for a complexity which is linear in the set of edges  $E$  we will use the notation  $O(E)$  in place of  $O(|E|)$ .

Section 2 gives some preliminary material on combinatorial auctions and graph matching. Section 3 is the longest section, it gives graphical structures for a wide variety of bids, and gives results on the type of bids which graph matching cannot handle. Section 4 provides additional results on the types of bids which lead to NP-equivalent auctions, and displays these results along with Section 3's results in graphical form. Section 5 gives complexity results for solving the identified classes, by using the current most efficient graph matching techniques available. Section 6 discusses related work and Section 7 concludes.

## 2. Preliminaries

This section describes the standard combinatorial auction problem, and the variant of it which we will be tackling. It then roughly describes how an optimal allocation can be computed by a reduction to graph matching, and surveys the main results in the relevant literature on 2D-graph matching algorithms.

### 2.1 Combinatorial Auctions

In a combinatorial auction the auctioneer is selling a set  $O = \{o_1, o_2, \dots, o_m\}$  of  $m$  objects to  $n$  potential buyers. A bundle of objects  $B$  is any subset of the objects, for example  $B = \{o_5, o_8, o_9\}$ . A bid is a pair  $(B, p)$  in which  $B$  is a bundle of objects and  $p$  is the valuation (or price offer) for the objects in  $B$ . Given a set  $\mathfrak{B}$  of  $k$  bids,  $\mathfrak{B} = \{(B_1, p_1), (B_2, p_2), \dots, (B_k, p_k)\}$  the combinatorial auction problem (CAP) is to find a set of bids which do not overlap, and which make the most money. Or, formally, to find a subset  $\mathfrak{B}_M \subseteq \mathfrak{B}$  such that  $\sum_{(B_i, p_i) \in \mathfrak{B}_M} p_i$  is maximal, under the constraint that for all  $(B_i, p_i), (B_j, p_j) \in \mathfrak{B}_M$  :  $B_i \cap B_j = \emptyset$ , meaning that the bundles in accepted bids cannot overlap. The allocation problem in this auction is what the literature refers to as the Combinatorial Auction Problem, or CAP. The assumption in this auction is that any non overlapping bids can be accepted; this corresponds to the OR bidding language. There is also an XOR bidding language where bidders may make mutually exclusive price offers for a number of bundles (i.e. "I will buy  $B_1$  or  $B_2$  but not both"). In such an auction each bid is a set,  $\{(B_1, p_1), (B_2, p_2), \dots, (B_k, p_k)\}$ , of bundles and price offers, where only one can be accepted. De Vries and Vohra (2003) call this the CAP2. This can be easily extended to the OR-of-XORs language where a bidder combines a number of XOR offers via an inclusive OR (Sandholm, 2002). In this case no modification is necessary to the CAP2 allocation algorithm, since we can treat each OR-of-XOR bid as a set of XOR bids; the fact that they

happen to be from the same bidder does not affect the allocation. OR-of-XORs describes the type of bidding in our graph based auctions. For most auctions we will assume that some objects can be left unallocated at the end of the auction.

A superadditive combinatorial auction is one where for all disjoint bundles  $B_i, B_j \subseteq O$ , an agent's price offer for  $B_i \cup B_j$  is greater than or equal to the sum of his price offers for  $B_i$  and  $B_j$ . In superadditive auctions we will say that there is a *surcharge* on a bundle. A subadditive combinatorial auction is one where for all disjoint bundles  $B_i, B_j \subseteq O$ , an agent's price offer for  $B_i \cup B_j$  is less than or equal to the sum of his price offers for  $B_i$  and  $B_j$ . In subadditive auctions we will say that there is a *discount* on a bundle. XOR bids are relevant in subadditive auctions, but they are redundant in superadditive auctions, because in a superadditive auction the auctioneer will not accept two disjoint bundles for a low price if he can accept the combined bundle for a higher price.

## 2.2 2D-Graph Matching Algorithms for Auctions

In an auction we have some objects for sale, and some bids which are placed on them. We can draw this as a diagram where we draw a point on the left for each bid and a point on the right for each object. We connect each bid to the objects it is bidding on by drawing lines; we can write the price offered as an annotation on the line. This diagram can be represented by a graph. A graph is a pair  $G = (V, E)$  where  $V$  is a set of vertices and  $E$  is a set of edges. An edge is a set of two vertices (this a 2D graph). We will deal with weighted graphs, where each edge is assigned an integer weight (representing the price offer). We use  $w(e)$  to denote the weight of the edge  $e$ . In a bipartite graph  $V$  can be divided in two:  $V = V_1 \cup V_2$  (often the vertices  $V_1$  are drawn on the left and  $V_2$  on the right), so that edges can only go from an element of  $V_1$  to an element of  $V_2$ . Many auctions can be solved by representing them as a bipartite graph where bids are represented in  $V_1$  and the objects for sale are represented in  $V_2$ . We want to match objects up with the bids which are offering to pay the most. A matching  $M$  in a graph is a subset of  $E$  where no vertex of  $V$  is covered (touched) by more than one edge  $e \in M$  (which can capture the requirement that no object is sold to two different bidders). In a bipartite graph it is easy to visualise a matching as matching vertices on the left with those on the right. A vertex is called matched if it is covered by the matching, otherwise it is *exposed* (representing an object that fails to sell, or a bid that loses). A maximum weight matching is a matching which achieves the best value for the sum of the weights of its edges. An optimal allocation in our auction graph is then given by a maximum weight matching of the graph. In some auctions we will need to use general (nonbipartite) graphs, for which the matching problem is slightly more complicated. We will make use of the complexity bounds for the best known algorithms for the graph matching problems we consider.

The problem of finding maximum matchings in graphs has a long history, resulting in highly efficient algorithms. The Kuhn-Munkres algorithm (Bondy & Murty, 1976, p. 87) can compute a bipartite weighted matching in  $O(n^3)$  time if the appropriate data structures are used. An excellent explanation of the basic algorithm is given by Taha (1997, p. 195) (this matching problem is known as “the assignment problem” in the Operations Research literature). The Kuhn-Munkres algorithm, and other graph matching algorithms, work by finding *alternating paths*. By *alternating path* we mean a path of even or odd length which

alternates between matched and unmatched edges. A path is just a sequence of unique vertices connected by edges. Clearly if one finds an odd alternating path where the two ends are not matched, then one can flip all the edges (matched becomes unmatched and unmatched becomes matched) in the path and achieve a matching which has one more edge than the previous (such a path is called an *augmenting path*). This process is repeated to find a maximal matching. Algorithms also work with the “dual” version of the problem, which is to find a *minimum weight cover*. A cover  $C$  assigns an integer value to each vertex so that for any edge  $e = \{u, v\}$ , the sum of the cover of its two vertices is at least as large as the weight of  $e$ ; i.e.  $C(u) + C(v) \geq w(e)$ . A minimum weight cover is a cover which achieves the smallest value for the sum of the covers of all vertices. The relationship between minimum weight cover  $C$  and a maximum weight matching  $M$  is that for any edge  $e$  in  $M$ , the sum of the cover of its two vertices is exactly the weight of  $e$ . It follows that the total weight of all edges in a maximum weight matching will equal the total of the minimum weight cover of all vertices in  $V$ .

The current best algorithm for calculating a maximum weight matching in a bipartite graph is given by Gabow and Tarjan (1989). It makes use of scaling, and runs in  $O(\sqrt{V}E \log(VN))$  time, where  $N$  is the heaviest edge in the graph. This gets close to the  $O(\sqrt{V}E)$  bound of the well known Hopcroft-Karp algorithm for unweighted bipartite maximum matching (Cormen, Leiserson, Rivest, & Stein, 2001, p. 696). For cases where weights are extremely small Gabow (1985) has sketched an  $O(\sqrt{V}E)$  algorithm for weighted matching. Although this theoretically works for  $N = O(1)$ , Gabow states that it is useful in practice for  $N = 1$  or  $2$ . Kao, Lam, Sung, and Ting (2001) have more recently published a decomposition algorithm for calculating a maximum weight bipartite matching, which outperforms Gabow and Tarjan’s 1989 algorithm in the case of low weights; to be precise, the case where  $W = o(E \log(VN))$  where  $W$  is the sum of the weights of all edges. Kao et al.’s algorithm runs in  $O(\sqrt{V}W)$  time. If the average edge weight is  $A$  then  $W = AE$ , and we can see that  $A$  would need to be very low for Kao et al.’s algorithm to be preferred. Suppose  $N \approx A$  and  $|V| = 2^{10}$ , then  $A$  should be  $\leq 13$ . As we are interested in auctions with fine grained bidding, agents will need to use large integers (e.g.  $10^6$ ) to express their bids, making Gabow and Tarjan’s algorithm preferred. However it should be pointed out that Kao et al.’s algorithm is very elegant and simple to implement. It would be interesting to see if scaling and decomposition could be combined to give a superior algorithm, for example by successively decomposing edges whose weight is in the range  $[2^{\lfloor \log N \rfloor}, N]$ . However, this seems not possible, as to acquire the information about which edges can be safely decomposed seems to require that the weighted matching algorithm be run on the subgraph of heavier edges.

For the bipartite case, Feder and Motwani (1991) have introduced a technique which can bring a speed up factor of  $\log(V^2/E)/\log V$ ; i.e. this factor takes a value in the range  $[0, 1]$  and this value can be multiplied by the time bound for a bipartite matching. This factor has in fact been included in Kao et al.’s paper, and they quote their bound as  $O(\sqrt{V}W \log(V^2/E)/\log V)$ . It is also applicable to Gabow and Tarjan’s algorithm. To get a feel for the speed up it may help to write it as  $2 - \frac{\log E}{\log V}$ . One can see that as  $E$  gets very large, approaching  $V^2$  (i.e. a fully connected graph), the factor goes towards zero. This is indeed a remarkable speed up for large highly connected graphs, and arouses some curiosity. The idea behind it is that a large highly connected graph will necessarily contain

large bipartite cliques; these can be compressed to increase efficiency, because finding a maximum matching in a bipartite clique is trivial, and when this is part of a bigger graph, any augmenting path can easily run through the clique, flipping edges. Feder and Motwani's compressed graph retains all the vertices of the original, but removes all the edges in each clique, and instead connects all these vertices to a new vertex which represents the clique. Because of these new added vertices, the compressed graph is tripartite. They then use Dinic's flow algorithm to find the maximum matching. Note that the time for computing the compression is negligible compared the time for computing the matching.

For maximum weight matching in general (nonbipartite) graphs the presence of blossoms (subgraphs containing odd loops) makes matching more difficult. Micali and Vazirani (1980) have given an  $O(\sqrt{V}E)$  algorithm for the unweighted case. Gabow and Tarjan (1991) tackle the weighted case using the technique of "shells" (introduced by Gabow) to tackle blossoms, giving an  $O(\sqrt{V\alpha(E, V)} \log V \cdot E \log(VN))$  time algorithm;  $\alpha$  is the inverse Ackermann function which grows extremely slowly. For all practical purposes  $\alpha(E, V) \leq 4$ , so we could consider it as a constant, giving a bound  $O(\sqrt{V \log V} \cdot E \log(VN))$  (the reality is only slightly larger).

In the above matchings each vertex has either one or zero matched edges incident to it (vertices at the ends of an edge are said to be *incident* to it, and vice versa); the weighted degree-constrained subgraph problem (DCS) generalises this. The input to the DCS problem is a multigraph (a graph where there can be multiple edges between any two vertices) where every vertex in the graph is assigned an  $l$ -value and a  $u$ -value (lower, upper) describing the minimum and maximum number of matched edges which can be incident to it in a DCS solution. Gabow and Tarjan (1991) note that a DCS problem, where the input multigraph has  $V$  vertices and  $E$  edges, can be reduced in linear time to a regular weighted matching problem on a new graph with  $O(E)$  vertices and edges (the construction of the new graph is given in Gabow (1985)). This gives a bound of  $O(\sqrt{\alpha(E, E)} \log E \cdot E^{3/2} \log(EN))$  for solving the weighted DCS. Gabow and Tarjan (1991) also mention that this can be improved with a careful implementation. This shows that DCS is merely a more concise way to represent some graph matching problems; it gives us no extra power to represent problems than normal graph matching. Tennenholtz (2002) reduces auctions to  $b$ -matching problems, a special case of DCS. In a  $b$ -matching every vertex in the graph is assigned a  $b$ -value describing the maximum number of matched edges which can be incident to it. There are two types of vertices in Tennenholtz's (2002)  $b$ -matching<sup>1</sup>, one has a fixed number of matched edges

---

1. We note that " $b$ -matching" has a number of different meanings, and that there is a lack of consistency in the literature when referring to different variants, thus the term can be quite confusing. Cook, Cunningham, Pulleyblank, and Schrijver (1998) describe " $(b, u)$ -matching" (*ibid.* page 182) and "maximum weight  $b$ -matching" (*ibid.* page 186), neither of which are the same as Tennenholtz's version. In Cook et al.'s " $(b, u)$ -matching" the vertices must have exactly  $b$  incident edges and  $u$ 's are capacities defined on edges. Cook et al.'s "maximum weight  $b$ -matching" requires that the number of incident edges must be  $\leq b$ , but there is no lower limit. Gabow (1983) calls this the upper degree constrained subgraph problem (UDCS). Jebara and Shchogolev (2006) use the same term as Cook et al., "maximum weight  $b$ -matching", to refer to a different problem: a matching with the requirement that the number of incident edges must be exactly  $b$ ; this is the problem which Cook et al. (*ibid.* page 182) call "perfect  $b$ -matching". When Müller-Hannemann and Schwartz (2000) refer to "weighted  $b$ -matching" they are talking about the same problem, i.e. the number of incident edges must be exactly  $b$  and there are no "capacities". In general we can say that Tennenholtz's use of the term " $b$ -matching" is unusual in that it refers to a problem with two distinct types of vertices; it is effectively like a combination of Cook et al.'s



$b$  incident to it, and the other accepts any number of matched incident edges in the range  $[0, b]$ .

It is also worth pointing out that all of the time bounds above are worst cases, and empirical evidence has shown that average case running times tend to be considerably better. Bast, Mehlhorn, Schäfer, and Tamaki (2004) have found average case results of  $O(E \log V)$  for the Hopcroft-Karp (unweighted bipartite) and Micali-Vazirani (unweighted nonbipartite) algorithms, whose worst case bounds were  $O(E\sqrt{V})$ . Since the weighted matching algorithms are based on these unweighted ones, similar speed ups can be expected. Bast et al.’s results come from randomly generated sparse graphs.

All of the matching algorithms mentioned above are for what we call 2D-graphs, where each edge connects exactly two vertices; hypergraphs generalise this, allowing an edge to connect  $n$  vertices. When we subsequently talk about 2D-matching techniques we are referring to all those algorithms mentioned above: bipartite matching, nonbipartite matching and the DCS.

### 3. Graphical Structures for Various Bids

This section shows how auctions with various different types of bids can be represented with graphs, and how the optimal allocations can be solved by using a graph matching algorithm. In general each auction is represented by a weighted bipartite graph  $G = (V_1 \cup V_2, E)$  where each bid is represented by a number of vertices in  $V_1$  and each object for sale is represented by a single vertex in  $V_2$ . The edges connecting bids to objects have a weight which is the price offered for that object. A singleton bid  $(\{o\}, p)$  (which means a price offer  $p$  for object  $o$ ) is represented by adding a single vertex to  $V_1$  to represent the bid, and an edge with weight  $p$  connecting to the vertex of  $V_2$  which represents object  $o$ . The bidding here implies an OR over all bids placed; i.e. one bidder can place a number of bids, and the auctioneer interprets this as an OR over all those bids. The graph will not be fully connected with  $E = V_1 \times V_2$  unless every bidder places a bid for every object; we are primarily interested in situations where the graph is reasonably sparse, this seems to be the typical case in the distributed AI scenarios which we are most interested in. For example, consider multi-agent systems implementing eCommerce applications where a large range of objects are for sale, and each agent is interested in a small subset of them. The optimal allocation can be calculated by finding a maximum weight matching in  $G$ . If all bids are singletons this is a simple matter of picking the largest bid placed on each object. We now describe a number of different types of nonsingleton bids which can all be combined in this type of auction. In all the descriptions below we assume  $V_2$  has one vertex corresponding to each object for sale. The additions to  $V_1$  for each bid type are described case by case. The earlier auctions have been identified as tractable by Tennenholtz, we point out differences where they arise.

---

“maximum weight  $b$ -matching” and “perfect  $b$ -matching”. Penn and Tennenholtz (2000) generalise this further and refer to “(general)  $b$ -matching”, allowing the lower limit on the number of incident edges to be in the range  $[0, b]$  (whereas above it is only allowed to be either zero or  $b$ ). Gabow (1983) calls this the degree constrained subgraph problem (DCS).

### 3.1 Bids for Multiple Objects with a Quantity Constraint

The quantity constrained bid is a bid which makes price offers for a number ( $k$ ) of objects, but puts a limit ( $q < k$ ) on the overall number which will be allocated. The price paid for any bundle  $B$  which is eventually allocated (where  $|B| \leq q$  objects) will be simply the sum of the prices offered on each object in  $B$ . Computing the optimal allocation is not trivial. The seller cannot simply accept the largest bid on each object. Suppose a bidder  $i$  makes an offer  $p_i$  for an object  $o_x$ , which exceeds the price any other bidder offers for that object; it does not necessarily follow that  $o_x$  will be allocated to bidder  $i$ . Suppose that the other bidders make reasonable offers for  $o_x$ , but there is some other object  $o_y$  for which they make very poor offers. Suppose further that  $i$  makes a good offer for  $o_y$ , then it may be optimal to accept  $i$ 's offer on  $o_y$  and to reject his large offer on  $o_x$ . Computing the optimal allocation is a non trivial combinatorial optimisation problem.

**Definition 3.1** A *Quantity-constrained multi-object bid (QCMOB)* is a subadditive combinatorial bid of the form  $(o_1, p_1, o_2, p_2, \dots, o_k, p_k, q)$  where each  $p_i$  is a price offer for object  $o_i$  and  $q$  is the maximum number of objects to be assigned to this bid. This represents an XOR over all  $\sum_{j=0}^q \binom{k}{j}$  possible selections of the  $k$  objects.

For each QCMO bid  $(o_1, p_1, o_2, p_2, \dots, o_k, p_k, q)$  we add  $q$  vertices to  $V_1$ . Let us call the vertices  $v_1, v_2, \dots, v_q$ . For each of these vertices  $v_i$ , we add  $k + 1 - i$  edges  $(v_i, o_i), (v_i, o_{i+1}) \dots (v_i, o_k)$  to  $E$ , connecting to objects  $o_i, o_{i+1}, \dots, o_k$ ; their weights are  $p_i, p_{i+1}, \dots, p_k$  respectively. The maximum weight matching in this graph gives the optimal allocation. Since each bid is represented by  $q$  vertices in  $V_1$ , it can only be allocated a maximum of  $q$  objects. Figure 1 illustrates a simple example. Notice that it is not necessary to connect each  $v_i$  to all  $k$  objects. By looking along the vertices  $v_1, v_2, \dots, v_q$  one can see that the total number of edges is  $k + (k - 1) + (k - 2) + \dots + (k - q + 1)$ . This is equal to the sum from 1 to  $k$  minus the sum from 1 to  $(k - q)$ , which gives  $\frac{k(k+1)}{2} - \frac{(k-q)(k-q+1)}{2} = kq - \frac{q^2-1}{2} = O(kq)$ , because  $k \geq q$ .

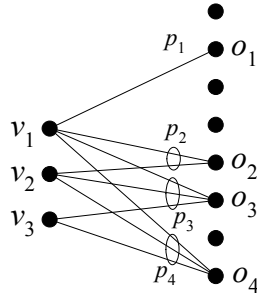


Figure 1: A QCMO bid  $(o_1, p_1, o_2, p_2, o_3, p_3, o_4, p_4, 3)$ . Offers are made for four objects but a maximum of three can be allocated.

The reason for this auction being called subadditive is that the bidder is expressing that his valuation for a bundle exceeding  $q$  objects is equal to his valuation for a subset of that bundle which contains  $q$  objects; hence he is willing to pay nothing for the extra objects, and his valuation on the larger bundle is less than his separate valuations on the bundle of size  $q$ , and the extra objects. The way in which we have represented the bid would not allow

the auctioneer to allocate extra objects to the bid (for free); if we want to allow this then we can add  $k - q$  dummy vertices on the left which connect to each object  $o_1, \dots, o_k$  and have weight zero. The use of maximum weight matchings would then give the following meaning to the bid  $b = (\dots, q)$ : the valuation for a bundle  $B$  with  $|B| > q$  is equal to the valuation for the subset of  $B$  of size  $q$  for which  $b$ 's valuation is maximal. In fact we can also come up with structures which give positive weight to these dummy links, meaning that specific subsequent objects can be bought at some discount, if they are purchased in conjunction with the other  $q$ . We will discuss the limitations of such structures further when we come to asymmetric bids (Section 3.7).

For QCMO bids our formulation is equivalent to Tennenholtz's, except that we use weighted matching in place of  $b$ -matching; this allows us to use a simpler matching algorithm. The  $b$ -matching formulation of many problems looks more concise, but as we have seen in Section 2.2, the maximum weight DCS algorithm will expand out all the vertices before running the normal graph matching algorithm. Penn and Tennenholtz (2000) have also defined the Partition-constrained multi-object auction (PCMOA) which is simply a collection of QCMO bids for each bidder. Each bidder partitions the objects into a collection of disjoint subsets, and places one QCMO bid on each subset.

### 3.2 Bids for Subadditive Pairs

A subadditive bid for a pair is simply a price offer for each individual object in the pair, along with a discounted valuation if the pair is to be allocated as a bundle.

**Definition 3.2** *A subadditive pair bid is a bid of the form  $(o_1, p_1, o_2, p_2, disc)$  which represents an XOR over the following offers:  $p_1$  is the price offered for object  $o_1$ ,  $p_2$  is the price offered for object  $o_2$  and  $(p_1 + p_2 - disc)$  is the price offered for the bundle  $\{o_1, o_2\}$ , where  $0 \leq disc \leq \min(p_1, p_2)$ .*

For each pair bid of the form  $(o_1, p_1, o_2, p_2, disc)$ , we add the following structure to our graph: Add one vertex  $p$  to  $V_1$  which offers each object at full price. This requires edges  $(p, o_1), (p, o_2)$  to be added to  $E$ , with weights  $p_1, p_2$  respectively. Add one vertex  $p_{-disc}$  to  $V_1$  which offers the second object at a discounted price. This requires edge  $(p_{-disc}, o_2)$  to be added to  $E$ , with weight  $p_2 - disc$ . Figure 2 illustrates this.

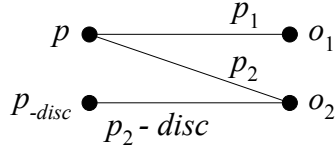


Figure 2: A pair bid  $(o_1, p_1, o_2, p_2, disc)$ . Offers are made for two objects but there is a discount  $disc$  if both are allocated. Objects  $o_1$  and  $o_2$  are represented on the right, and  $disc$  is the discount for purchasing both together.

Since the maximum weight matching algorithm is trying to maximise total weight, it will always pick the connection to  $p$  if a single object is allocated, hence guaranteeing the full price for a single object. This formulation is again equivalent Tennenholtz's, except for

the use of weighted matching in place of  $b$ -matching. The constraint  $disc \leq \min(p_1, p_2)$  is necessary to avoid having a negative weight edge to vertex  $p_{-disc}$  (which would never be matched). This seems a reasonable constraint for the auction, as to violate it would effectively mean that, having bought one object, the bidder would be paid to take the second one away; on the other hand one could imagine a scenario where some negative synergy exists between the objects (for example exceeding the threshold for some tax) such that being allocated the second is a burden. Allowing the negative weight edge would allow this bid to be expressed, but it is not really meaningful if the matching never takes it. To make it meaningful we would have to have a “forced sale”. We remove the auctioneer’s free disposal and force certain objects (i.e. all those for which a  $disc > \min(p_1, p_2)$  bid has been placed) to be allocated by setting their degree to be 1, and using a DCS in place of normal matching.

We now explain an alternative equivalent structure to represent binary bids. This structure gives no advantages in the pair case; it has one extra edge, and one extra vertex. However, it will prove useful in the later auctions, and it is convenient to introduce it now on this simple example. In this alternative structure, for each pair bid  $(o_1, p_1, o_2, p_2, disc)$ , we add one vertex  $d$  to  $V_2$  (this is known as a discount vertex) and add two vertices  $v_1, v_2$  to  $V_1$ , and for each of  $v_1, v_2$ , add two edges, e.g. for  $v_1$  add  $(v_1, o_1), (v_1, d)$  with weights  $p_1, disc$  respectively, to  $E$ . Figure 3 illustrates this.

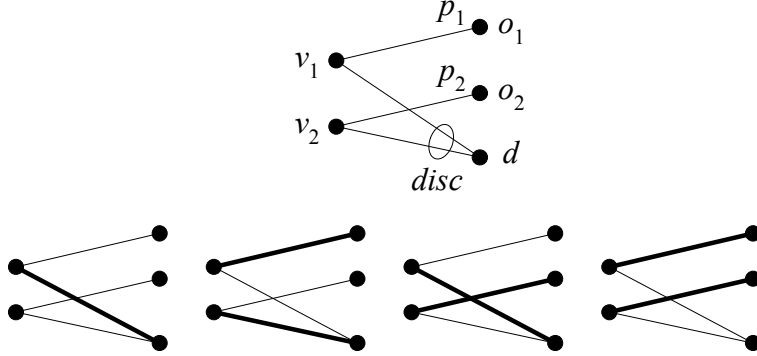


Figure 3: Alternative bipartite structure to model a pair bid  $(o_1, p_1, o_2, p_2, disc)$  (top). The four lower diagrams indicate the matchings (heavy lines) corresponding to buying (i) no objects; (ii) object  $o_1$ ; (iii) object  $o_2$ ; (iv) objects  $o_1$  and  $o_2$ .

Now without any object being purchased this structure contributes a weight  $disc$  to the overall maximum weight matching. Therefore we subtract  $disc$  from the final value of the matching. If both objects are purchased by this bid, then the  $disc$  edge will be lost, giving the required discount to the bidder. We still need the constraint  $disc \leq \min(p_1, p_2)$  as without it the cheaper object would never be allocated. The maximum weight matching in a graph including these structures gives the optimal allocation. The price to be paid by each bidder is given by the sum of the weight of all that bidder’s edges which are included in the matching (i.e. all the matched edges of the structures corresponding to his bids), minus the  $disc$  of each of his pair bids. The total value to the auctioneer is given by the

value of the maximum weight matching minus the *disc* of each pair bid. Again we could get over the constraint  $disc \leq \min(p_1, p_2)$  if we use a forced sale (using DCS).

### 3.3 Almost Additive Bids

Almost additive bids are another type of bid which Tennenholtz has handled, and which we include for completeness. In this bid individual price offers are placed on  $k$  objects. If  $< k$  objects are allocated then the price paid is simply the sum of the individual prices. If the whole  $k$  are allocated then there is a discount.

**Definition 3.3** *An almost additive bid is a bid of the form  $(o_1, p_1, o_2, p_2, \dots, o_k, p_k, disc)$  which represents an XOR over the following offers: the price offer for any strict subset  $S \subset \{o_1, \dots, o_k\}$  is  $\sum_{o_i \in S} p_i$  and the price offer for the full set is  $(\sum_{i=1}^k p_i) - disc$ . We require that  $0 \leq disc \leq p_i$  for all  $i$ .*

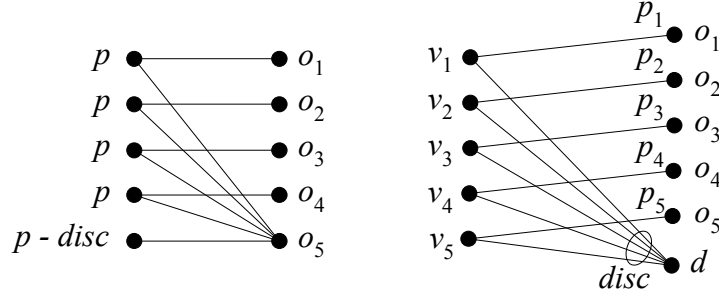


Figure 4: An almost additive bid  $(o_1, p_1, o_2, p_2, \dots, o_k, p_k, disc)$  for  $k = 5$ . Offers are made for  $k$  objects but there is a discount  $disc$  if all  $k$  objects are allocated. The “standard” structure is on the left and the “alternative” on the right. Edge weights have not been included on the left, but they should be obvious by analogy with the pair case (see Figure 3).

In Figure 4 we show both the “standard” and “alternative” structures which can represent an almost additive bid  $(o_1, p_1, o_2, p_2, \dots, o_k, p_k, disc)$ . We will forego a detailed description as it should be obvious by extension of the pair structures (see previous section). In the standard structure each object allocated will take the most expensive edge available, so the discount vertex will only be matched if there is no other one available; i.e. if the whole set is being allocated. In the alternative structure the  $d$  vertex will be matched as long as fewer than  $k$  objects are being allocated, because it then has a free  $v_i$  vertex to match; only if the whole set is being allocated does it need to be broken. If no object is allocated the alternative structure still generates revenue  $disc$  so this must be subtracted from the overall revenue (similar to the pair case). Again our formulation is equivalent to Tennenholtz’s except that weighted matching makes things considerably simpler than  $b$ -matching

### 3.4 Bids for Subadditive Triples

The case of triples is where things really start to get interesting. We know that combinatorial auctions with triples are in general NP-hard, but we will see that the special case of subadditive symmetric triples is tractable.

**Definition 3.4** *A subadditive symmetric triple bid is a bid of the form  $(o_1, p_1, o_2, p_2, o_3, p_3, d_2, d_3)$  which represents an XOR over the following offers:  $p_i$  is the price offered for object  $o_i$ , and  $(p_i + p_j - d_2)$  is the price offered for any pair bundle  $\{o_i, o_j\}$ , and  $(p_1 + p_2 + p_3 - d_3)$  is the price offered for the whole triple bundle  $\{o_1, o_2, o_3\}$ . Given that we are talking about subadditive bids, discounts  $d_2, d_3$  must be positive.*

(Read  $d_2$  as “discount for a pair” and  $d_3$  as “discount for a triple”.) Symmetric here (and in all subsequent tuple bids) means that the discount for any subset of the tuple is the same, e.g. in the triple case the discounts for  $\{o_1, o_2\}$  and  $\{o_1, o_3\}$  and  $\{o_2, o_3\}$  are all the same. To handle subadditive symmetric triple bids we need to employ different graphical structures depending on the relative magnitudes of the discounts  $d_2, d_3$ .

#### 3.4.1 THE CASE WHERE $d_3 \geq 2d_2$

**Theorem 1** *Combinatorial auctions with subadditive symmetric triple bids of the form  $(o_1, p_1, o_2, p_2, o_3, p_3, d_2, d_3)$  where  $d_3 \geq 2d_2$  and  $p_i > d_3 - d_2$  (for all  $i$ ) are tractable and can be solved by bipartite matching.*

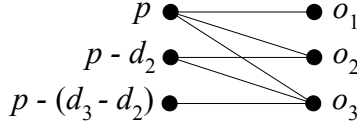


Figure 5: A triple bid  $(o_1, p_1, o_2, p_2, o_3, p_3, d_2, d_3)$ . Offers are made for three objects, at three different prices. The most expensive ( $p$ ) will be taken if only one object is allocated; if a second is allocated it cannot connect to  $p$  as it has been used up, so it must connect to the cheaper  $p - d_2$ ; likewise the third will have to connect to the even cheaper  $p - (d_3 - d_2)$ . Edge weights have not been included, but they should be obvious by analogy with the pair case (see Figure 3).

**Proof.** Again we will give both the standard and alternative structures to represent the triple bid  $(o_1, p_1, o_2, p_2, o_3, p_3, d_2, d_3)$ . For the standard version we add three vertices to  $V_1$  (Figure 5 illustrates this): the first vertex offers each object at full price, the second vertex offers the second object  $i$  at the pair discount price  $p_i - d_2$ , the third offers the third object  $i$  at the triple discount price  $p_i - (d_3 - d_2)$ . Clearly we need the third offer to be no more attractive than the second (otherwise it would have been taken first), therefore we need  $p_i - (d_3 - d_2) \leq p_i - d_2$  which gives us the constraint  $d_3 \geq 2d_2$ . It can be seen that the discount for the third object “gives back” the discount for the second; thus if all three are allocated the total price is  $p_1 + p_2 + p_3 - d_2 - (d_3 - d_2) = (p_1 + p_2 + p_3 - d_3)$ , as desired. ■

Figure 6 shows the alternative version. If no object is allocated the structure still generates revenue  $d_2 + (d_3 - d_2) = d_3$  so this must be subtracted from the overall revenue

(similar to the pair case). If a pair is allocated the cheaper link ( $d_2$ ) will be broken, leading to the desired reduction in revenue. If all three are allocated both links will be broken, and since there is an overall reduction of  $d_3$  in the revenue, the overall value will be correct. The alternative version has five extra (beyond the objects) vertices where the standard version has three. For general  $n$ -tuples the alternative version has  $2n - 1$  vertices and  $\frac{n(n+1)}{2} + n - 1$  edges where the standard has  $n$  vertices and  $\frac{n(n+1)}{2}$  edges. In both cases there are  $O(n)$  vertices and  $O(n^2)$  edges. We will not use the standard version from now on, as we will see the advantages of the alternative in the next section.

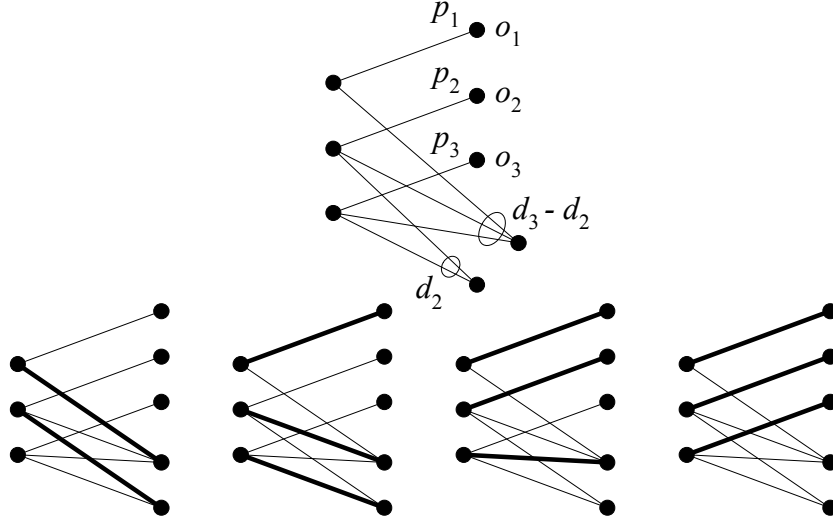


Figure 6: Alternative bipartite structure to model a triple bid  $(o_1, p_1, o_2, p_2, o_3, p_3, d_2, d_3)$  (top). The four lower diagrams indicate matchings (heavy lines) corresponding to buying (i)  $\{\}$ ; (ii)  $\{o_1\}$ ; (iii)  $\{o_1, o_2\}$ ; (iv)  $\{o_1, o_2, o_3\}$ .

Tennenholtz (2002) handles triple bids which satisfy the constraint  $d_3 > 3d_2$ . We have relaxed this constraint and thus shown that a larger class is tractable; we will relax it further in the next subsection.

### 3.4.2 THE CASE WHERE $d_2 \leq d_3 < 2d_2$

We encounter an interesting problem when we try to make the discount  $d_3$  less than twice  $d_2$ . Because the discount for a triple must include the pair discount as well, the “extra” discount for the triple is  $d_3 - d_2$ , which starts to become less than  $d_2$  as  $d_3$  goes less than  $2d_2$ . This means that the matching algorithm will prefer to take the  $d_3 - d_2$  discount first. We can view the matching algorithm as being the auctioneer who is trying to maximise revenue. If a pair is to be allocated he will take the link giving the bigger revenue, i.e. the link with discount  $d_3 - d_2$  (bigger revenue=smaller discount), which is not the valuation the bidder has given for a pair. This leads us to wonder if there might be some clever bipartite structure  $S$  which can represent the bidder’s valuation for a pair, and force the matching algorithm to take the bigger discount first. The following theorem will prove that there is not. (Note that there follows a long sequence of lemmas and theorems, and the reader who is

more interested in possibilities than impossibilities could skip to Figure 8 and Theorem 6 at this stage.) We work with the assumption that our bipartite structure  $S$  (which represents a bid) would be connected by edges to the objects (in the main graph) on which the bid is placing offers (example  $S$  structures appear in all the preceding figures). Thus to find the value of the bid, we consider the maximum weight matching of the graph formed from  $S$  and the vertices corresponding to whatever objects eventually get allocated to this bid. We can view the allocation or non-allocation of an object to this bid as the inclusion or removal of an object vertex in  $S$ . To handle the discount  $d_3 < 2d_2$  we need a structure that gives a bigger increase in value if an object  $o_k$  is added to structure  $S + \{o_i, o_j\}$  than if it is added to structure  $S + \{o_i\}$ . We will show that this is impossible, and more generally it is impossible for the increase in value, due to adding any object  $o$  after some others, to be greater than if  $o$  were added before. We will eventually generalise to DCS over multigraphs to show that even that does not help. We first need a lemma due to Kao, Lam, Sung, and Ting (2000).

**Lemma 1 (Kao et al. (2000))** *Let  $G = (V, E)$  be a weighted graph and let  $M$  be a maximum weight matching of the graph. For any vertex  $v \in V$ :*

1. *If  $v$  is not matched by  $M$ , then  $M$  is also a maximum weight matching of the graph  $G - \{v\}$ .*
2. *If  $v$  is matched by  $M$ , then  $G$  contains a single alternating path  $P$  starting from  $v$  which can transform  $M$  to a maximum weight matching of the graph  $G - \{v\}$ .*

**Proof.** (following Kao et al. (2000)) Statement 1 is straightforward. For 2: let  $M_{-v}$  be a maximum weight matching of  $G - \{v\}$ . Now consider the symmetric difference  $M \cup M_{-v} - M \cap M_{-v}$  (all the edges in one but not the other), this must be a set  $A$  of vertex disjoint alternating paths and cycles. One path must end on  $v$ , because  $v$  is matched in  $M$  and not in  $M_{-v}$ ; call this path  $P$ . Now suppose we transformed the original matching  $M$  by just  $P$ ; we would have a matching  $M'_{-v}$ . We will now show that  $M'_{-v}$  must in fact be a maximum weight matching; any other paths and cycles in  $A$  do not improve the value. The other paths and cycles in  $A$  do not touch  $v$ , therefore they could just as well be applied to change  $M$ ; if they cause an increase in the value of the matching then  $M$  was not maximal originally and we reach a contradiction. ■

We can also show that this works backwards.

**Lemma 2** *Let  $G = (V, E)$  be a weighted graph. For any vertex  $v \in V$ , let  $M_{-v}$  be a maximum weight matching of the graph  $G - \{v\}$ :  $G$  contains a single alternating path  $P$  starting from  $v$  which can transform  $M_{-v}$  to a maximum weight matching of  $G$ .*

(We don't have two cases this time, because  $P$  could have length zero, corresponding to the case where  $v$  remains unmatched in the maximum weight matching of  $G$ )

**Proof.** Firstly note that the proof does not immediately follow from Lemma 1. Let  $M$  be a maximum weight matching of  $G$ . Lemma 1 tells us that there is an alternating path  $P'$  which can convert  $M$  to a maximum weight matching of  $G - \{v\}$ . However  $G - \{v\}$  might have several maximum weight matchings; Lemma 1 only guarantees that at least one of them can be transformed to  $M$  by  $P'$ . Given that  $M_{-v}$  is one of the maximum weight



matchings of  $G - \{v\}$  it is possible that it is one which cannot be extended to a maximum weight matching of  $G$  by a single alternating path  $P$  starting at  $v$ . Suppose we have a non extendible  $M_{-v}$ ; its symmetric difference from  $M$  is a set  $A$  of vertex disjoint alternating paths and cycles. At most one path may end on  $v$ , that is if  $v$  is matched in  $M$  and not in  $M_{-v}$ ; call this path  $P$  (if  $v$  is not matched in  $M$  then make  $P$  a dummy path of length zero). Now suppose we transform  $M_{-v}$  by just  $P$ ; we would have a matching  $M'$  which must in fact be a maximum weight matching of  $G$ ; any other paths and cycles in  $A$  do not improve the value. The other paths and cycles in  $A$  do not touch  $v$ , therefore they could just as well be applied to change  $M_{-v}$ ; if they cause an increase in value then  $M_{-v}$  was not a maximum weight matching of  $G - \{v\}$  and we reach a contradiction. Therefore any maximum weight  $M_{-v}$  must be extendible. ■

Note that this works equally well for non-bipartite graphs; the proof did not make use of the bipartite property of a graph. It also works for the DCS (degree constrained subgraph) provided that the vertex being added has degree one ( $u = 1$ ). We generalise it to handle the other cases.

**Lemma 3** *Let  $G = (V, E)$  be a weighted multigraph. For any vertex  $v \in V$ , with upper degree  $u$ , let  $M_{-v}$  be a maximum weight DCS of the multigraph  $G - \{v\}$ :  $G$  contains at most  $u$  alternating paths  $P_1, \dots, P_u$  starting from  $v$  which can transform  $M_{-v}$  to a maximum weight DCS of  $G$ .*

**Proof sketch.** (Similar to the previous cases) Add the edge connections one by one (following the routine used to add vertices above). The added alternating paths may interact, but the difference between  $M_{-v}$  and the final DCS  $M$  must be a set of alternating paths and cycles, and any paths or cycles which do not contact  $v$  could have been applied to change  $M_{-v}$ , so they must not cause any improvement. This leaves us with the paths which contact  $v$ , of which there are at most  $u$ . ■

We can generalise this further.

**Lemma 4** *Let  $G = (V, E)$  be a weighted multigraph. For any set of vertices  $v_1 \dots v_k \in V$ , with upper degrees  $u_1 \dots u_k$ , let  $M_{-v}$  be a maximum weight DCS of the multigraph  $G - \{v_1 \dots v_k\}$ :  $G$  contains at most  $\sum_{i=1}^k u_i$  alternating paths  $P_j$ , with at most  $u_i$  alternating paths starting from each  $v_i$ , which can transform  $M_{-v}$  to a maximum weight DCS of  $G$ .*

**Proof.** Obvious, given the previous proofs. ■

Now we present the main theorem. Essentially it states that adding  $o_2$  first always leads to a better increase than adding it after  $o_1$  (or at least an equal increase). Let  $\text{mwm}(G)$  denote any maximum weight matching of  $G$ , and let  $w(\text{mwm}(G))$  denote its weight.

**Theorem 2** *Given a bipartite graph  $G = (V_1 \cup V_2, E)$  and two new vertices  $o_1$  and  $o_2$  which are to be added to  $V_2$  (along with any edge connections from  $\{o_1, o_2\}$  to  $V_1$ ):*  
 $w(\text{mwm}(G + \{o_2\})) - w(\text{mwm}(G)) \geq w(\text{mwm}(G + \{o_1, o_2\})) - w(\text{mwm}(G + \{o_1\}))$ .

**Proof.** Let  $P_1$  be the alternating path used to transform  $\text{mwm}(G)$  to  $\text{mwm}(G + \{o_1\})$ . Let  $P_2$  be the alternating path used to transform  $\text{mwm}(G + \{o_1\})$  to  $\text{mwm}(G + \{o_1, o_2\})$ . If  $P_1$  and  $P_2$  do not interact then we are done;  $P_2$  could just as well be applied to  $\text{mwm}(G)$  to

give the exact same increase in value, so we have equality. Assume  $P_1$  and  $P_2$  do interact. This interaction cannot be in the form of simply touching at a vertex; they must have overlapping edges. To see this note that both are alternating paths, every vertex on the path is matched (except possibly the last) if they touch at a vertex then two matched edges are touching, which is not permitted. The possibility of the last vertex touching is open; we have two cases: either the last vertices of both paths meet, or the last vertex of one path meets the somewhere in middle of the other. An allowed meeting must mean that the last edge on one path is unmatched. If the last edge  $e_i$  of one path  $P_i$  meets the middle of the other  $P_j$ , it must be that  $e_i$  is an unmatched edge, but this means that the application of  $P_i$  changed a previously matched edge at  $e_i$  into an unmatched one. This would mean that before the application of  $P_i$ , a matched edge was touching the middle of  $P_j$  which is impossible (this case closed). Now we tackle the case where both ends meet. The bipartite property of the graph means that when either path is taking an unmatched edge it is going from  $V_1$  to  $V_2$  (recall that both paths start in  $V_2$ ). Now if both ends are to meet it must be in  $V_2$  when both are ending with unmatched edges (if they met in  $V_1$  they would both be matched). This means that  $P_2$  is not dependent on  $P_1$  having been applied first, it could just as well have been applied to change  $\text{mwm}(G)$  (this case closed). Now we have established that the interaction takes the form of overlapping edges.  $P_2$  may join and leave  $P_1$  multiple times, overlapping with some edges each time. However neither path crosses over itself (that would not be a path, but would include a cycle; Lemma 2 guarantees no cycles). We will define “upstream” to mean the a part of  $P_1$  which is closer to  $o_1$ ; “downstream” shall mean farther from  $o_1$ . Now let us consider the farthest downstream interaction (overlapping) between  $P_1$  and  $P_2$ .  $P_2$  must connect to  $P_1$  via a matched edge of  $P_2$ ; the next edge  $e$  of  $P_2$  is an unmatched edge of  $P_2$  which inverts what had been a matched edge of  $P_1$ , and  $P_2$  must be travelling upstream (see example in Figure 7). Let  $e = \{d, u\}$  where  $d$  is the

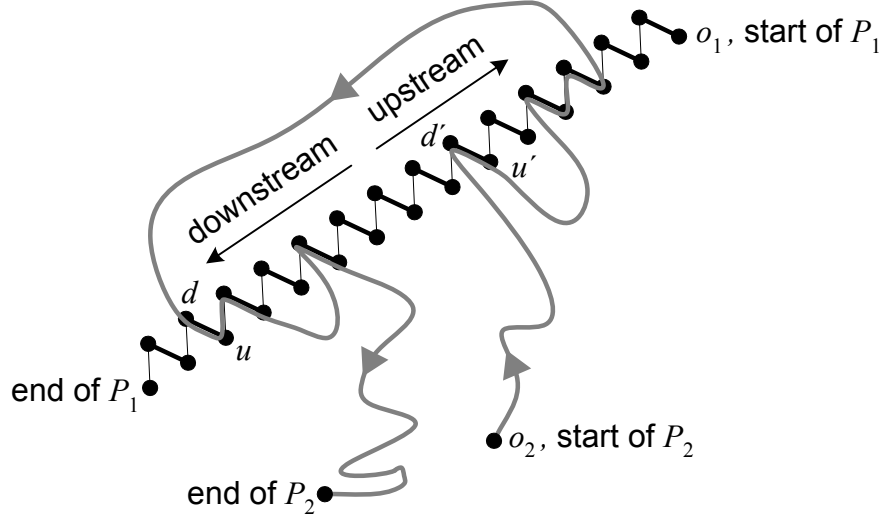


Figure 7: Interaction between alternating path  $P_1$  (black) and  $P_2$  (grey).  $P_2$  has not been applied to the matching yet, but its path is shown.

downstream vertex and  $u$  is the upstream vertex. Now whatever  $P_2$  does after  $u$  causes a

certain change in the value of the matching, call this  $\Delta_2$ . Let  $\Delta_1$  be the change in the value of the matching which had been brought about by the portion of  $P_1$  from  $u$  downstream to the end of  $P_1$ .  $\Delta_2$  cannot exceed  $\Delta_1$  because otherwise  $P_1$  would have not followed that path (to maximise value), it would instead have followed  $P_2$  from  $u$ ; note that this holds true even if  $P_2$  subsequently overlaps with more sections of  $P_1$  further upstream; this would correspond to  $P_1$  undoing some of what it had done, and if it leaves and rejoins it forms a cycle (which has no value by Lemma 2). Having established that  $\Delta_2$  cannot exceed  $\Delta_1$ , we can see that if we want to apply  $P_2$  to change  $\text{mwm}(G)$ ,  $P_2$  could have done just as well or better if it had followed  $P_1$  downstream instead of  $P_2$  from the vertex  $d$ . Note that the value of  $P_2$  from  $d$  onwards is  $\Delta_2 - w(e)$  whereas the value of  $P_1$  from  $d$  downstream is  $\Delta_1 - w(e)$ , and  $\Delta_1 - w(e) \geq \Delta_2 - w(e)$ . Thus we have constructed a portion of an alternating path which can be used to add  $o_2$  to  $G$ , and generate at least as much value as  $P_2$  does when adding  $o_2$  to  $G + \{o_1\}$ . Now strip away all of  $P_2$  from  $u$  onwards (i.e. invert its edges to make things as they were before it was applied), then follow  $P_1$  upstream from  $u$  until the next place where  $P_2$  joins into it (it may be that  $P_2$  overlaps  $P_1$  for a few edges before leaving, and rejoining). Notice that this next overlapping must be an earlier (closer to  $o_2$ ) portion of  $P_2$ . Apply the same procedure again. Let  $e'$  be the next overlapped edge, with  $e' = \{d', u'\}$ .  $P_2$  travels on some path from  $u'$  to  $d$ ; clearly this portion of  $P_2$  cannot have a greater value than the portion of  $P_1$  which goes from  $u'$  to  $d$  (otherwise  $P_1$  should have taken that to maximise). Therefore the portion of  $P_1$  which goes from  $u'$  to  $d$  gives us a portion of a path which could be taken to add  $o_2$  to  $G$  and obtain an increase in value at least as good as that obtained when  $P_2$  is applied to  $\text{mwm}(G + \{o_1\})$ . Clearly we can keep continuing the procedure upstream along  $P_1$  until we reach the overlapping closest to  $o_1$ . Now we have a complete path which can be used to add  $o_2$  to  $G$  to obtain an increase in value at least as good as that obtained when  $o_2$  is added to  $G + \{o_1\}$  via  $P_2$  being applied to  $\text{mwm}(G + \{o_1\})$ . The complete path is: start at  $o_2$ , follow  $P_2$  until the first place where it meets  $P_1$ , then follow  $P_1$  downstream to its end. ■

We now present a couple of generalisations of this. We could have presented the most general theorem directly, but given the length of the above proof, it probably aids readability to present things incrementally.

**Theorem 3** *Given a bipartite graph  $G = (V_1 \cup V_2, E)$  and new set of vertices  $O_1$  and a single vertex  $o_2$  which are to be added to  $V_2$  (along with any edge connections from  $O_1 \cup \{o_2\}$  to  $V_1$ ):  $w(\text{mwm}(G + \{o_2\})) - w(\text{mwm}(G)) \geq w(\text{mwm}(G + O_1 \cup \{o_2\})) - w(\text{mwm}(G + O_1))$ .*

**Proof.** By Lemma 3,  $\text{mwm}(G + O_1)$  adds a set  $\mathcal{P}_1$  of vertex disjoint alternating paths to  $\text{mwm}(G)$ ;  $\text{mwm}(G + O_1 \cup \{o_2\})$  adds a further alternating path  $P_j$  on top of this. Following the approach in the proof of Theorem 2, we can take each path  $P_i$  in  $\mathcal{P}_1$  one by one, and follow them from their downstream end upstream until they contact  $O_1$ . We need to be careful to take each path  $P_i$  in  $\mathcal{P}_1$  in a particular order: we start with the last one that was added (note that Lemma 3 tells us that adding a vertex later cannot be better than adding it earlier, but it could be worse, so order is important). Now let  $P_i$  be the last path of  $\mathcal{P}_1$  that was added; we start following it upstream. Each time we come to a vertex  $v$  where path  $P_j$  meets  $P_i$ , we then can say that the value increase which was caused by the portion of  $P_i$  from  $v$  downstream is at least as good as what  $P_j$  gets after  $v$ .

Otherwise  $P_i$  should have followed  $P_j$  from  $v$  instead; this is a feasible path for  $P_i$  because  $P_i$  is the last path of  $\mathcal{P}_1$  that was added, hence any subsequent interactions which  $P_j$  has with  $\mathcal{P}_1 \setminus \{P_i\}$  could also have been performed by  $P_i$ ; there are no subsequent downstream crossings of  $P_i$ , nor any subsequent crossings of  $P_j$  (because  $P_j$  is a path, which is vertex disjoint by definition). Now we strip away (invert edges) the portion of  $P_j$  from  $v$  onwards, and continue scanning upstream along  $P_i$  until the next interaction. Thus we eventually get to the furthest upstream interaction between  $P_i$  and  $P_j$ , let us say that the downstream end of this overlapped portion is vertex  $d$ ; we now know that by adding  $o_2$  to  $G$ , and using  $P_j$  as far as  $d$  and following  $P_i$  downstream thereafter, we improve the value of  $w(\text{mwm}(G))$  by at least as much as  $w(\text{mwm}(G + O_1 \cup \{o_2\})) - w(\text{mwm}(G + O_1))$ . We then progress to the second last path  $P_h$  of  $\mathcal{P}_1$  which had been added. Note that there are no further interactions between  $P_j$  and  $P_i$ ; we have stripped away all of these parts of  $P_j$ . Therefore we can safely apply the same “tracing upstream procedure” on  $P_h$ ; each time we meet an interaction at a vertex  $u$  we know that the portion of  $P_j$  from  $u$  to  $d$  can only interact with a further upstream part of  $P_h$ , or paths which were added before  $P_h$  (and hence which had been available for  $P_h$  to interact with had it wanted to). We continue tracing upstream on each path of  $\mathcal{P}_1$ , in reverse order of how they were added, until we end up at a vertex  $j$  which is the first place where  $P_j$  interacts with any path of  $\mathcal{P}_1$ ; say this interaction is on path  $P_a \in \mathcal{P}_1$ . We can now guarantee that by adding  $o_2$  to  $G$ , and using  $P_j$  as far as  $j$  and following  $P_a$  downstream thereafter, we improve the value of  $w(\text{mwm}(G))$  by at least as much as  $w(\text{mwm}(G + O_1 \cup \{o_2\})) - w(\text{mwm}(G + O_1))$ . ■

Let  $\text{mwd}(G)$  denote any maximum weight DCS of  $G$ , and let  $w(\text{mwd}(G))$  denote its weight.

**Theorem 4** *Given a bipartite multigraph  $G = (V_1 \cup V_2, E)$  and two new sets of vertices  $O_1$  and  $O_2$  which are to be added to  $V_2$  (along with any edge connections from  $O_1 \cup O_2$  to  $V_1$ ):  $w(\text{mwd}(G + O_2)) - w(\text{mwd}(G)) \geq w(\text{mwd}(G + O_1 \cup O_2)) - w(\text{mwd}(G + O_1))$ .*

**Proof.** (by induction on the size of  $O_2$ ) Theorem 3 gives us our base case, i.e. when the size of  $O_2$  is one. For the inductive case we need to show that if

$$w(\text{mwd}(G + O_2)) - w(\text{mwd}(G)) \geq w(\text{mwd}(G + O_1 \cup O_2)) - w(\text{mwd}(G + O_1))$$

holds for  $O_2 = O_n$  where  $O_n$  is some set of vertices of size  $n$ , then it will also hold for a larger  $O_2 = O_n \cup \{o_{n+1}\}$  where  $o_{n+1}$  is any extra vertex we add in. If we can prove this then we know that it holds for any  $O_2$  because given any  $O_2$  we can simply build it up by adding the vertices one by one. The assumption of our inductive step is

$$w(\text{mwd}(G + O_n)) - w(\text{mwd}(G)) \geq w(\text{mwd}(G + O_1 \cup O_n)) - w(\text{mwd}(G + O_1))$$

We want to show that this assumption implies that

$$\begin{aligned} & w(\text{mwd}(G + O_n \cup \{o_{n+1}\})) - w(\text{mwd}(G)) \\ & \geq w(\text{mwd}(G + O_1 \cup O_n \cup \{o_{n+1}\})) - w(\text{mwd}(G + O_1)) \end{aligned} \quad (3.1)$$

By Theorem 3 we have

$$\begin{aligned} & w(\text{mwd}(G + \{o_{n+1}\})) - w(\text{mwd}(G)) \\ & \geq w(\text{mwd}(G + O_1 \cup \{o_{n+1}\})) - w(\text{mwd}(G + O_1)) \end{aligned} \quad (3.2)$$

and if we make a graph out of  $G + \{o_{n+1}\}$ , our assumption gives us

$$\begin{aligned} & w(\text{mwd}((G + \{o_{n+1}\}) + O_n)) - w(\text{mwd}(G + \{o_{n+1}\})) \\ & \geq w(\text{mwd}((G + \{o_{n+1}\}) + O_1 \cup O_n)) - w(\text{mwd}((G + \{o_{n+1}\}) + O_1)) \end{aligned} \quad (3.3)$$

Note that  $w(\text{mwd}((G + \{o_{n+1}\}) + O_1))$  must be the same as  $w(\text{mwd}(G + O_1 \cup \{o_{n+1}\}))$ . The DCSs might not be the same, but their weights must be (otherwise one is not maximal). Adding both sides of the inequalities 3.2 and 3.3 gives us what we want (inequality 3.1). ■

**Theorem 5** *Assume that a combinatorial auction problem is represented by a bipartite multigraph  $G = (V_1 \cup V_2, E)$ , and that the objects for sale are each represented by a vertex in  $V_2$ , and that bids are represented by any bipartite structures at all, and that the value of the optimal allocation is proportional to the weight of a maximum weight DCS. Combinatorial auctions with subadditive symmetric triple bids of the form  $(o_1, p_1, o_2, p_2, o_3, p_3, d_2, d_3)$  where  $d_2 \leq d_3 < 2d_2$  and  $p_i > d_3 - d_2$  (for all  $i$ ) cannot be solved by finding a maximum weight DCS of  $G$ .*

**Proof.** If  $d_3 < 2d_2$  then the gain in value for allocating object  $o_i$  as the second of a pair will be less than the gain in value if that same object is allocated as the third of a triple. The allocation of an object entails an increase in value, hence some edges must be added to connect to the object. To correctly model our auction we require that the connection of the object as the second of a pair must bring less gain in value than if that same object is connected as the third of a triple (i.e. after another one has been connected). Theorem 4 shows that this is not possible. ■

The Theorem does not rule out the possibility that bipartite graphs might handle the problem if each object were represented by vertices in both  $V_1$  and  $V_2$ . This would allow direct interaction between two bids. However it is not clear how the structure would be arranged so that an object is either fully allocated or not, and a partially allocated object would not correspond to a feasible allocation in the auction the graph is supposed to represent. More outlandish schemes might also be possible, for example if the value of an allocation were not proportional to the weight of a maximum weight DCS, but instead represented by some particular relationship among edges in a maximum weight DCS. Thus we cannot claim that it is impossible to solve the auction as a DCS for *any* encoding scheme.

Note that the bipartite property of the graph was used in the first part of the proof of Theorem 2. If we move to nonbipartite graphs we open one little hole in the proof, and this hole can be exploited to handle triple bids with  $d_3 < 2d_2$ . The hole which is opened is the case where both ends of the alternating paths for  $o_1$  and  $o_2$  meet. Figure 8 illustrates the appropriate structure. There is a single nonbipartite link with weight  $d_2$  which gives back the pair discount if the triple is allocated.

**Theorem 6** *Combinatorial auctions with subadditive symmetric triple bids of the form  $(o_1, p_1, o_2, p_2, o_3, p_3, d_2, d_3)$  where  $d_3 \geq d_2$  and  $p_i > d_3$  (for all  $i$ ) are tractable and can be solved by nonbipartite matching.*

**Proof.** See Figure 8. Note that if no object is allocated the structure still generates revenue  $d_2 + d_3$  so this must be subtracted from the overall revenue (similar to the pair case). ■

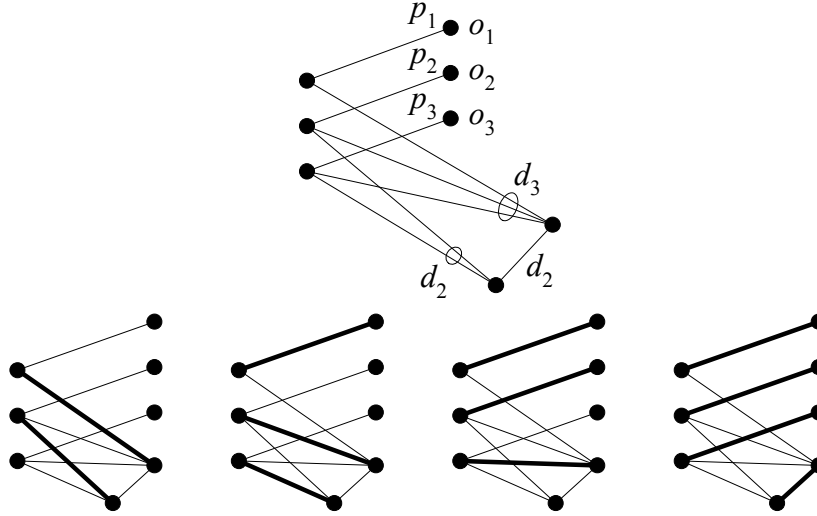


Figure 8: Nonbipartite structure to model a triple bid  $(o_1, p_1, o_2, p_2, o_3, p_3, d_2, d_3)$  (top). The four lower diagrams indicate matchings (heavy lines) corresponding to buying (i)  $\{\}$ ; (ii)  $\{o_1\}$ ; (iii)  $\{o_1, o_2\}$ ; (iv)  $\{o_1, o_2, o_3\}$ .

Note that the condition  $p_i > d_3$  is actually no more restrictive than the condition  $p_i > d_3 - d_2$  in Theorem 1, because Theorem 1 also had the condition  $d_3 \geq 2d_2$ , whereas now we have  $d_3 \geq d_2$ . If we want very small  $p_i$  values, the smallest we can make them is  $d_2$  in both cases.

**Corollary 1** *Given a nonbipartite multigraph  $G = (V_1 \cup V_2, E)$  and two new sets of vertices  $O_1$  and  $O_2$  which are to be added to  $V_2$  (along with any edge connections from  $O_1 \cup O_2$  to  $V_1$ ): The only way to achieve  $w(\text{mwd}(G + O_2)) - w(\text{mwd}(G)) < w(\text{mwd}(G + O_1 \cup O_2)) - w(\text{mwd}(G + O_1))$  is if some of the ends of the paths added due to  $O_2$  touch some of the ends of the paths added due to  $O_1$ .*

**Proof.** It can be seen from the proofs of Theorems 2 to 4 that the bipartite property was used only once (in Theorem 2) and that was to guarantee that paths cannot gain advantage by meeting end to end; the remainder of all proofs work for the nonbipartite case. ■

Note also that the use of DCSs does not help at all; therefore in future proofs we will not constantly refer to DCSs, but simply prove some limitations for graph matching, knowing that they also apply to DCSs.

### 3.4.3 THE CASE WHERE $d_3 < d_2$

This case is actually superadditive because the valuation on a pair plus the valuation on the excluded singleton is less than the valuation on the triple. We will see when we come to superadditive bids that we cannot handle this case with our matching techniques.

### 3.5 Bids for Subadditive Quadruples

Tennenholtz (2000) mentions the possibility of extending his technique to bundles of larger size, but says “The conditions ... become more elaborated, which might make the results less applicable.” By using our weighted matching structures in place of Tennenholtz’s  $b$ -matching approach, we do not suffer the same constraints.

**Definition 3.5** *A subadditive symmetric quadruple bid is a bid of the form  $(o_1, p_1, o_2, p_2, o_3, p_3, o_4, p_4, d_2, d_3, d_4)$  which represents an XOR over the following offers:  $p_i$  is the price offered for object  $o_i$ , and  $(p_i + p_j - d_2)$  is the price offered for any pair  $\{o_i, o_j\}$ ,  $(p_i + p_j + p_k - d_3)$  is the price offered for any triple  $\{o_i, o_j, o_k\}$ , and  $(p_1 + p_2 + p_3 + p_4 - d_4)$  is the price offered for the whole quadruple. Given that we are talking about subadditive bids, discounts  $d_2, d_3, d_4$  must be positive.*

Again, we need to employ different graphical structures depending on the relative magnitudes of the discounts.

#### 3.5.1 THE CASE WHERE $d_4 \geq 2d_3 - d_2$ AND $d_3 \geq 2d_2$

**Theorem 7** *Combinatorial auctions with subadditive symmetric quadruple bids of the form  $(o_1, p_1, o_2, p_2, o_3, p_3, o_4, p_4, d_2, d_3, d_4)$  where  $d_4 \geq 2d_3 - d_2$  and  $d_3 \geq 2d_2$  and  $p_i > d_4 - d_3$  (for all  $i$ ) are tractable and can be solved by bipartite matching.*

**Proof.** See Figure 9 (left). We know from Theorem 1 that we need  $d_3 \geq 2d_2$  to make sure the third discount is not taken second. To make sure the fourth discount is not taken third we also need  $d_4 - d_3 \geq d_3 - d_2$  which gives us  $d_4 \geq 2d_3 - d_2$ . Note that it would not be safe to write the constraint as  $d_4 \geq 3d_2$  except in the case where  $d_3 = 2d_2$ . If no object is allocated the structure still generates revenue  $d_4$  so this must be subtracted from the overall revenue. ■

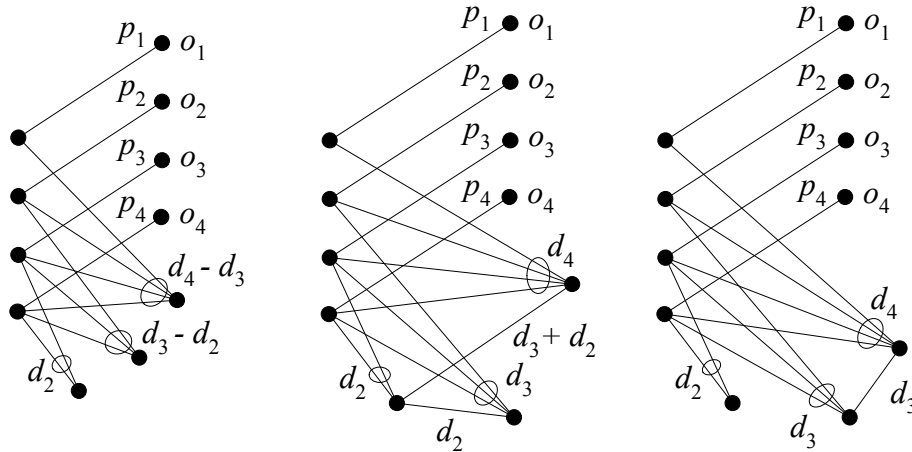


Figure 9: Bipartite structure to model a quadruple bid (left). Nonbipartite structures to model a quadruple bid (right and centre).

### 3.5.2 THE CASE WHERE $d_3 \geq d_2$ AND $d_4 \geq d_3 + d_2$

**Theorem 8** *Combinatorial auctions with discounted symmetric quadruple bids of the form  $(o_1, p_1, o_2, p_2, o_3, p_3, o_4, p_4, d_2, d_3, d_4)$  where  $d_3 \geq d_2$  and  $d_4 \geq d_3 + d_2$  and  $p_i > d_4$  (for all  $i$ ) are tractable and can be solved by nonbipartite matching.*

**Proof.** See Figure 9 (centre). If no object is allocated the structure still generates revenue  $d_2 + d_3 + d_4$  so this must be subtracted from the overall revenue. ■

Note that nonbipartite matching brings us a limited element of superadditivity here. Let us make the discount for purchasing a third object ( $d_3 - d_2$ ) and the discount for purchasing a fourth object ( $d_4 - d_3$ ) as small as possible; this happens when  $d_3 = d_2$  and  $d_4 = d_3 + d_2 = 2d_2$ . Now the third object is allocated at full price, as is the fourth. Consider the value of any pair  $\{o_i, o_j\}$  and the remaining pair  $\{o_k, o_l\}$ ; these values are  $p_i + p_j - d_2$  and  $p_k + p_l - d_2$ , totalling  $p_1 + p_2 + p_3 + p_4 - 2d_2$ . The value of the quadruple is  $p_1 + p_2 + p_3 + p_4 - d_2$ . Provided  $d_2 > 0$  this is superadditivity. However the limitation is that we can only use our superadditivity once. In Figure 9 (centre) we have made our structure so that the increase in value for allocating an object fourth cannot exceed the increase for allocating it third. See that the discount for the third can be less than the second (i.e.  $d_3 < 2d_2$  and the discount on the third allocation is  $d_3 - d_2$ ), but the discount for the fourth can be no less than  $d_3 - d_2$ .

### 3.5.3 THE CASE WHERE $d_4 \geq d_3 > 2d_2$

We could alternatively have changed the structure as shown in Figure 9 (right). Then the discount for the fourth could be less than the discount for the third ( $d_3 \leq d_4 < 2d_3$ , the  $d_4 < 2d_3$  is not a constraint on this structure, we are just describing what is possible), but the third must be greater than the second ( $d_3 > 2d_2$ ). This turns out not to be superadditive, because the value of a quadruple does not exceed the value of a triple and a singleton; however in an  $n$ -tuple, a reduced discount for the fourth of many could make superadditivity. Note that there is always the limitation that we cannot have two small discounts in sequence.

**Theorem 9** *Combinatorial auctions with discounted symmetric quadruple bids of the form  $(o_1, p_1, o_2, p_2, o_3, p_3, o_4, p_4, d_2, d_3, d_4)$  where  $d_4 \geq d_3 > 2d_2$  and  $p_i > d_4$  (for all  $i$ ) are tractable and can be solved by nonbipartite matching.*

**Proof.** See Figure 9 (right). If no object is allocated the structure still generates revenue  $d_2 + d_3 + d_4$  so this must be subtracted from the overall revenue. ■

## 3.6 Bids for Subadditive $n$ -Tuples

By now it is easy to see how the structures generalise to  $n$ -tuples.

**Definition 3.6** *A discounted symmetric  $n$ -tuple bid is a bid of the form  $(o_1, p_1, \dots, o_n, p_n, d_2, \dots, d_n)$  which represents an XOR over the following offers: for any bundle  $B \subseteq \{o_1, \dots, o_n\}$ ,  $(\sum_{o_i \in B} p_i) - d_{|B|}$  is the price offered for  $B$ . Each singleton  $\{o_i\}$  must have offer  $p_i$  so we fix  $d_1 = 0$ . Discounts  $d_2, \dots, d_n$  must be positive.*



**Theorem 10** *Combinatorial auctions with discounted symmetric  $n$ -tuple bids of the form  $(o_1, p_1, \dots, o_n, p_n, d_2, \dots, d_n)$  where  $d_j \geq 2d_{j-1} - d_{j-2}$  for all  $j = 3 \dots n$ , and  $p_i > d_n - d_{n-1}$  (for all  $i$ ) are tractable and can be solved by bipartite matching.*

**Proof.** The appropriate structure is the extension of Figure 9 (left) to handle  $n$  objects. If no object is allocated the structure still generates revenue  $d_n$  so this must be subtracted from the overall revenue. ■

We can see now that that the almost additive bids of Section 3.3 are a special case where  $d_2 \dots d_{n-1}$  are zero. Furthermore, we can generalise the above bids further so that QCMO bids become a special case of them.

**Definition 3.7** *A quantity constrained discounted symmetric  $n$ -tuple bid is a bid of the form  $(o_1, p_1, \dots, o_n, p_n, q, d_2, \dots, d_q)$ , with  $q \leq n$ , which represents an XOR over the following offers: for any bundle  $B \subseteq \{o_1, \dots, o_n\}$  with  $|B| \leq q$ ,  $(\sum_{o_i \in B} p_i) - d_{|B|}$  is the price offered for  $B$ . Each singleton  $\{o_i\}$  must have offer  $p_i$  so we fix  $d_1 = 0$ . Discounts  $d_2, \dots, d_q$  must be positive.*

**Theorem 11** *Combinatorial auctions with quantity constrained discounted symmetric  $n$ -tuple bids of the form  $(o_1, p_1, \dots, o_n, p_n, q, d_2, \dots, d_q)$  where  $d_j \geq 2d_{j-1} - d_{j-2}$  for all  $j = 3 \dots q$ , and  $p_i > d_q - d_{q-1}$  (for all  $i$ ) are tractable and can be solved by bipartite matching.*

**Proof.** The appropriate structure is built by first treating it as a normal discounted symmetric  $n$ -tuple bid for the objects  $o_1, \dots, o_q$ , and constructing the structure as Theorem 10 would dictate. Then to handle the extra objects  $o_{q+1}, \dots, o_n$  we must add edges just as in the QCMO structure of Section 3.1; i.e. for each of the vertices  $v_i \in V_1$  which belong to this bid, we have  $n + 1 - i$  edges  $(v_i, o_i), (v_i, o_{i+1}) \dots (v_i, o_n)$ , connecting to objects  $o_i, o_{i+1}, \dots, o_n$ ; their weights are  $p_i, p_{i+1}, \dots, p_n$  respectively. If no object is allocated the structure still generates revenue  $d_q$  so this must be subtracted from the overall revenue. ■

Now QCMO bids are the special case where all discounts are zero. Discounted symmetric  $n$ -tuple bids are the special case where  $q = n$ . We now look at general  $n$ -tuples where discounts may be decreasing.

**Theorem 12** *Combinatorial auctions with quantity constrained discounted symmetric  $n$ -tuple bids of the form  $(o_1, p_1, \dots, o_n, p_n, q, d_2, \dots, d_q)$  where  $d_j \geq \sum_{k=2}^{j-1} d_k$  for all  $j = 3 \dots q$ , and  $p_i > d_q$  (for all  $i$ ) are tractable and can be solved by nonbipartite matching.*

**Proof.** The appropriate structure is the extension of Figure 9 (centre) to handle  $q$  objects, and then extra edges to handle the extra objects  $o_{q+1}, \dots, o_n$ , as described in Theorem 11 above. If no object is allocated the structure still generates revenue  $\sum_{k=2}^q d_k$  so this must be subtracted from the overall revenue. ■

This is one simple structure that generalises easily, but more arrangements are possible. With this structure the discounts can only increase after the third allocation. A decreased discount is allowed only on the third allocation. We can shift the place where a decreased discount can happen further down the line by rearranging the structure, for example copying the ideas of Figure 10 (centre and right). However we will always suffer limitations similar to those discussed in the superadditive case (Section 3.8). Decreasing discounts (just like

superadditivity) can only come in pairs, we can never have two in sequence; the best we can do is: big discount, little discount, big discount, little discount, ... where “big” means at least twice the previous one.

### 3.7 Asymmetric Subadditive Bids

We have tended to focus on symmetric discounts because there is no general concise formula we can give for asymmetric  $n$ -tuple bids; asymmetric means that we are intending to put bespoke discounts on each singleton, pair, etc. This gives a lot of possibilities for a large  $n$ -tuple when compared with the symmetric case which was limited to assigning discounts for each of the  $n - 1$  symmetric possibilities. Let us firstly focus on the triple case to see what is possible. The allocation of the whole triple can only have one value, so there is no symmetry issue to discuss; the interesting cases are the pairs. In a triple  $\{o_1, o_2, o_3\}$  there are three possible pairs:  $\{o_1, o_2\}$  and  $\{o_2, o_3\}$  and  $\{o_1, o_3\}$ . Suppose each is to have a unique discount:  $d_{12}, d_{23}, d_{13}$  with  $d_{12} > d_{23} > d_{13}$ . Now each singleton  $o_i$  is offered at price  $p_i$ , so there must be an alternating path by which each  $o_i$  can obtain this value. If any one of these paths is taken it must disable the others, so that they cannot be subsequently allocated at full price. Paths may end by making an edge matched (when it was unmatched before) or unmatched (when it was matched before). A path which disables something must end by making an edge matched (note that every vertex on a path is matched, except possibly the last, which is not matched iff the path ends by making an edge unmatched). Now given that  $o_1$  is allocated we want to be able to allocate  $o_3$  for a price  $o_3 - d_{13}$ , therefore we need a separate path for  $o_3$  which offers this discount (we could have chosen to make this separate path for  $o_1$ , it is an equivalent alternative). Alternatively, suppose that  $o_3$  is allocated first, we then want to be able to allocate  $o_2$  for a price  $o_2 - d_{23}$ , therefore we need a separate path for  $o_2$  which offers this discount (we could have chosen to make this separate path for  $o_1$ , it is an equivalent alternative). Now we have the appropriate discounts for the pairs  $\{o_2, o_3\}$  and  $\{o_1, o_3\}$ . However if we want to allocate  $\{o_1, o_2\}$ , our matching algorithm will use the singleton path allocation for  $o_1$  and the  $o_2 - d_{23}$  discounted path for  $o_2$ , giving us a total value  $o_1 + o_2 - d_{23}$ . This is greater than the  $o_1 + o_2 - d_{12}$  discounted value we desired. Our  $d_{12}$  discount is not achievable. The fundamental limitation is that we cannot create a discounted path for  $o_2$  which is available *only if*  $o_3$  has been already allocated;  $o_3$  is itself a disabling path, not an enabling one. Graph matching can only match or unmatch edges; it cannot disable one thing and enable another. Thus when we created the discounted path to achieve  $d_{23}$ , we in fact made a path for object  $o_2$ , it did not capture a relationship among objects  $o_2$  and  $o_3$ .

In an  $n$ -tuple, there will be  $n(n - 1)/2$  pairs, and every time we make a discounted path for an object  $o_i$  we will in fact be making a path which allows any pair with  $o_i$  to be allocated. Let us create discounted paths for pairs in order of smallest discount first. When we create the first path (for some object  $o_i$ ), we determine the discount for  $n - 1$  pairs (i.e. all pairs which contain this object), when we create the second path (for some other object  $o_j$ ) we determine the discount for  $n - 2$  pairs (because  $\{o_i, o_j\}$  already has a better value), and so on ( $n - 3, n - 4 \dots$ ) until one is left ( $n - (n - 1) = 1$ ). We have created  $n - 1$  paths. Therefore of the  $n(n - 1)/2$  possible pairs, we can assign unique discounts to only  $n - 1$ .

Generalising further, an  $n$ -tuple contains  $\binom{n}{k}$  possible subsets of size  $k$ . Of these, how many can have a unique discount? It is certainly no more than  $n$  as we create discounts by creating paths for up to  $n$  objects; however  $n$  is not achievable because (as above) we run out of unique elements to use for a discounted path. The maximum number of unique elements will be given by the size of the smallest hitting set of the subsets. This number is  $n - k + 1$ . To see this note that when we make the first discounted path we are left with  $\binom{n-1}{k}$  subsets which do not use the object we just made a path for. After the second we are left with  $\binom{n-2}{k}$  subsets and so on until the top becomes  $k$  and we are left with one subset. This happens at the  $(n - k)$ th step, after which we can make one more unique discount, giving  $n - k + 1$ .

**Theorem 13** *In a combinatorial auction, for any bid on a set of  $n$  objects, the maximum number of nonempty nonsingleton subsets over which XOR price offers can be placed is  $2^n - (n + 1)$ . A subadditive auction which is solvable by graph matching (or DCS), can give a unique discount to a maximum of  $n(n - 1)/2$  of these.*

**Proof.**  $2^n - (n + 1)$  is the total number of subsets less the singletons and the empty allocation. For all other subsets of any size  $k$ , the maximum number of subsets of size  $k$  for which a unique discount can be placed is  $n - k + 1$  (see paragraph above Theorem). Summing these up for  $k = 2 \dots n$  gives  $n(n - 1)/2$ . ■

Note that this applies to QCMO bids as well as  $n$ -tuples.

### 3.8 Superadditive Bids

We have seen a limited degree of superadditivity in our quadruple and  $n$ -tuple bids already, but there we never had an allocated object exceeding its individual price; i.e. the value increase due to allocating an extra object onto a bundle never exceeded the value of that object allocated as a singleton. This would be a stronger type of superadditivity. Corollary 1 has shown us that there is only one way that we can get a greater value increase by adding an object  $o_2$  after an object (or collection)  $o_1$ , than the singleton value of  $o_2$ . That one way is to use nonbipartite matching, and to make the tail ends of the alternating paths for  $o_1$  and  $o_2$  meet. Exploiting this method we can do a superadditive pair bid, as shown in Figure 10 (left).

**Definition 3.8** *A superadditive pair bid is a bid of the form  $(o_1, p_1, o_2, p_2, c_2)$  which represents an XOR over the following offers:  $p_1$  is the price offered for object  $o_1$ ,  $p_2$  is the price offered for object  $o_2$  and  $(p_1 + p_2 + c_2)$  is the price offered for the bundle  $\{o_1, o_2\}$ , where  $c_2 > 0$ .*

(Read  $c_2$  as the surcharge for a pair.) Note that if we solved this as an OR auction rather than an XOR the allocation would be the same, because the auctioneer will never allocate the two as singletons if the pair value is greater (this applies in all superadditive cases).

**Theorem 14** *Combinatorial auctions with superadditive pair bids of the form  $(o_1, p_1, o_2, p_2, c_2)$  where  $c_2 > 0$  are tractable and can be solved by nonbipartite matching.*

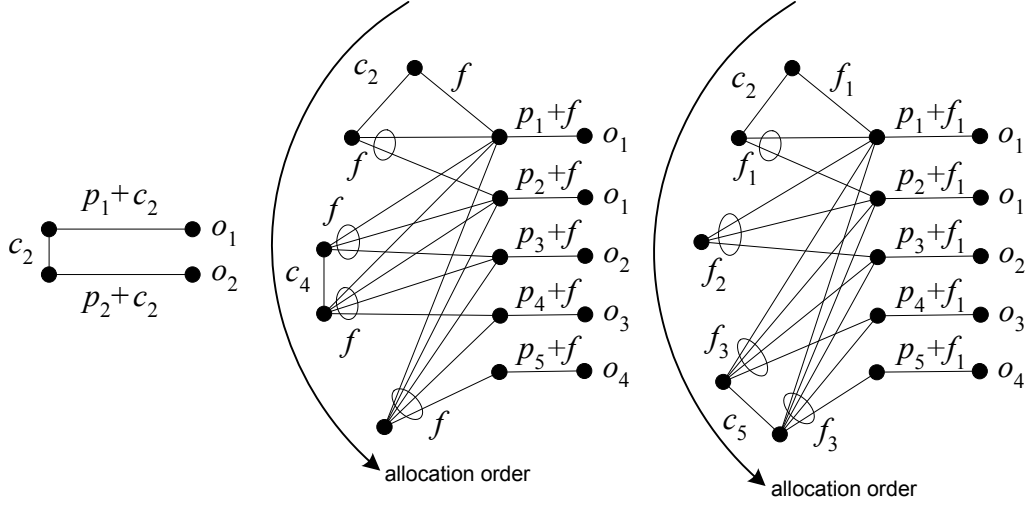


Figure 10: Nonbipartite structure to model a superadditive pair bid (left). Nonbipartite structure to model a quintuple bid with no discounts (centre) and with discounts (right).

**Proof.** For each pair bid of the form  $(o_1, p_1, o_2, p_2, c_2)$ , we add the structure of Figure 10 (left) to our graph. If no object is allocated the structure still generates revenue  $c_2$  so this must be subtracted from the overall revenue. We can see that if one object is allocated the  $c_2$  link is broken, losing  $c_2$  revenue, but the object link costs  $p_i + c_2$  so the net gain is  $p_j$  as desired. If a second is allocated the increase in value is  $p_i + c_2$ , as desired. ■

We now present a theorem to show that structures for superadditivity over pairs do not extend to  $n$ -tuples as nicely as they did in the subadditive case. Firstly let us formalise our notion of symmetry:

**Definition 3.9** A symmetric  $n$ -tuple bid represents an XOR over price offers for every subset of the bundle, where the price offers satisfy the following constraints. An individual value is expressed for each singleton in the  $n$ -tuple, but for any two (not necessarily disjoint) subsets  $S_1$  and  $S_2$  where  $|S_1| = |S_2|$ , the value difference  $v_1 = v(S_1) - \sum_{s \in S_1} v(s)$  must equal  $v_2 = v(S_2) - \sum_{s \in S_2} v(s)$ ; where the  $v(\cdot)$  function gives the value placed on a subset of the  $n$ -tuple. It can be seen that a symmetric  $n$ -tuple bid can be completely described by a tuple  $(o_1, p_1, \dots, o_n, p_n, \delta_2, \dots, \delta_n)$ , giving the  $n$  objects and their singleton prices and a sequence  $\delta_2, \dots, \delta_n$  of  $n - 1$  discounts or surcharges (let us generically refer to these as deltas, with a positive delta being a surcharge and a negative one being a discount). There must be no delta for the allocation of a singleton, therefore we say  $\delta_1 = 0$ .

The type of superadditivity we are interested in is where the tuple is selling for more than the sum of its singletons; this requires that some object is allocated for more than its singleton value, i.e. there is some  $\delta_i$  for  $i \in [2, n]$  such that  $\delta_i - \delta_{i-1} > 0$ . For convenience let us define  $inc_i$  to be the increase in value brought by the  $i$ th allocation, not counting the singleton value of the object allocated; i.e.  $inc_i = \delta_i - \delta_{i-1}$ .

**Theorem 15** *In a symmetric auction which is solvable by graph matching, for any  $n$ -tuple bid of the form  $(o_1, p_1, \dots, o_n, p_n, \delta_2, \dots, \delta_n)$ :*

1. *For all  $i \in [1, n]$ :  $inc_i > 0$  implies  $inc_{i-1} \leq 0$*
2. *Each positive pair brings a smaller increment than all previous ones; i.e. for all  $i \in [2, n]$ :  $inc_i > 0$  implies that for all  $j \in [1, i-1]$ :  $inc_j + inc_{j-1} \geq inc_i + inc_{i-1}$*

**Proof.** For Part 1: Corollary 1 has shown that the only way to achieve more than the singleton price for an object  $i$  is if the alternating path added for its allocation meets end on end with some earlier path. This means that the allocation of some earlier object has “set-up”<sup>2</sup> a path which the  $i$ th allocation can exploit. That earlier allocation must be the  $(i-1)$ th. This can be proven by observing that each “set-up” path must be exploited immediately by the next allocation. Let the  $q$ th allocation be the first to set-up a path, this is then available to be exploited by *any* subsequent object allocation (because of symmetry, all objects have equal ability to get deltas). Then the  $(q+1)$ th allocation must exploit it by taking the most expensive edge that touches the end of  $q$ s path, because after each object added to the allocation a maximum weight matching is computed, and no other path will give  $(q+1)$  a value greater than this. Observe also that the  $q$ th allocation cannot generate more value than its singleton value, this follows because we have said that it is the first to set-up a path. When the  $(q+1)$ th allocation exploits  $q$ s path, it has used the most expensive edge that touched the end of  $q$ s path, so there is no subsequent path that can touch this end to get more value. Therefore no subsequent allocation can be worth more than its singleton value unless there is another allocation which performs another set-up. Thus we have established that every allocation  $i$  with  $inc_i > 0$  must immediately follow an allocation  $(i-1)$  with  $inc_{i-1} \leq 0$ .

For Part 2: If  $inc_i$  is positive then object  $i$  used a path set-up by  $i-1$ , and  $i-1$ , being a set-up a path, cannot have exploited any previous set-up. Now (by symmetry) all objects have equal status, and equal access to exploit paths, and we know from Corollary 1 that the pair of paths taken by objects  $(i-1, i)$  would be worth at least as much if they had been taken before other objects instead of after; this is because the paths  $(i-1, i)$  meet end to end, so their ends are not free to touch any other paths. Therefore the paths which  $(i-1, i)$  took were available to any allocations  $(j-1, j)$  for all  $j \in [1, i-1]$ , and would have been taken, if they were worth more. Therefore the later pair  $(i-1, i)$  is worth at most the same as any earlier pair  $(j-1, j)$ . ■

**Corollary 2** *The only way for a delta to exceed all previous deltas is if it is part of a sequence starting at the first delta and ending at some index  $k$ , and having the following properties:*

1. *For each even  $i \in [4, k]$ :  $\delta_i > \delta_{i-2}$  and  $inc_i \leq inc_{i-2}$ .*
2. *For each odd  $i \in [3, k]$ :  $inc_i \leq 0$  and  $inc_i \leq inc_{i-2}$ .*

---

2. We will use “set-up” with this specific meaning, i.e. the end of one alternating path has removed an edge from the matching, and that removed edge touched a high weight edge which is now free to be matched by a subsequent path.

(i.e. the even deltas are monotonically increasing, the odd ones are at best static)

**Proof.** For Part 1: The first delta ( $\delta_1$ ) is zero, so to get a delta that exceeds all previous ones requires that it be positive, which can only be achieved if  $inc_i$  is positive for some  $i$ . This applies for all  $i \in [2, k]$ ; to get a  $\delta_i$  that exceeds all previous ones requires that  $inc_i$  is positive. Theorem 15 shows us that positive  $inc$ s come in pairs, where the first is a set-up, and the second reaps the increment. This shows that we can achieve a sequence as indicated. To show that there can be no break in the sequence, suppose that we have some  $i$  so that  $\delta_i$  exceeds all previous deltas (implying  $inc_i + inc_{i-1} > 0$ ) but which is not part of such a sequence; i.e. there is some previous even  $j$  with  $\delta_j \leq \delta_{j-2}$ . If  $\delta_{j-2}$  is lower than  $\delta_j$  then this means that allocation  $j$  and its set-up  $j-1$  have not brought any increment. Therefore  $inc_j + inc_{j-1} \leq 0 < inc_i + inc_{i-1}$  which violates Theorem 15 Part 2. The claim  $inc_i \leq inc_{i-2}$  is a consequence of Theorem 15 Part 2: for a consecutive sequence of three allocations we have  $inc_{i-1} + inc_{i-2} \geq inc_i + inc_{i-1}$ .

For Part 2:  $inc_i \leq 0$  follows from the proof of Part 1 because each odd allocation must be a set-up for the subsequent even one, and set-ups cannot exploit any edge at the end of their paths. Similar to the even case, the claim  $inc_i \leq inc_{i-2}$  follows because if a later set-up has a lower discount, it would have been taken earlier. ■

**Corollary 3** *The only way for a delta to exceed zero is if it is part of a sequence starting at the first delta and ending at some index  $k$ , and having the following properties:*

- For each even  $i \in [4, k]$ :  $\delta_i > 0$  and  $inc_i \leq inc_{i-2}$ .
- For each odd  $i \in [3, k]$ :  $inc_i \leq 0$  and  $inc_i \leq inc_{i-2}$ .

**Proof.** If delta dips below zero at any even  $i$ , then no subsequent pair can bring it above zero. Say  $i$  is the first even index where  $\delta_i < 0$ , this means  $inc_i + inc_{i-1} < 0$ , and to bring delta above zero a subsequent  $inc_k + inc_{k-1}$  would need to exceed zero, which violates Theorem 15. ■

This is a severe limitation on symmetric bids; if we want deltas to increase we can only do it in a very constrained pairwise fashion, i.e. each even numbered allocation bringing an increase. We show two possible schemes for symmetric superadditive bids over a quintuple in Figure 10 (centre and right). In the central structure we see the symmetric bid  $(o_1, p_1, \dots, o_5, p_5, 0, c_2, c_2, c_2 + c_4, c_2 + c_4)$ . This is an example with monotonically increasing deltas; this is the maximum that can be got from a quintuple. The constraints are:  $f > c_2 \geq c_4$ ;  $c_2$  can be anything. If no object is allocated the structure still generates revenue  $5f$ , so this must be subtracted from the overall revenue. In the right hand structure we see the following symmetric bid:

$$(o_1, p_1, \dots, o_5, p_5, 0, c_2, c_2 - (f_2 - f_1), c_2 - (f_2 + f_3 - 2f_1), c_2 + c_5 - (f_2 + 2f_3 - 3f_1))$$

The increments corresponding to these deltas are (starting with  $inc_1$ ):  $+c_2, -(f_2 - f_1), -(f_3 - f_1), -(f_3 - f_1 - c_5)$ . This is an example where we have pushed the second increase onto the fifth allocation, by the use of discounts. The constraints are:  $f_3 > f_2 > f_1 > c_2 \geq c_4$  and  $f_3 > c_5$  and  $p_i > f_2 - f_1$  (for all  $i$ );  $c_2$  can be anything, but if it is large then subsequent

discounts will need to be larger. If no object is allocated the structure still generates revenue  $2(f_1 + f_3) + f_5$ , so this must be subtracted from the overall revenue. Note that it is pointless to make the  $f$  values on either side of a surcharge different; whatever is taken away by the second  $f$  will be given back by the surcharge, so one may as well make the surcharge less instead. One further point to note is that the allocation order indicated in Figure 10 (right) is not the only possibility; if  $c_5 > f_3 - f_2$  then  $f_2$  will be used in the third allocation, but then released in favour of  $c_5$  for the fourth allocation, and then  $f_2$  will be taken again for the fifth allocation. It should be easy to see how the structures in these figures generalise to  $n$ -tuples.

**Definition 3.10** *A general super/subadditive symmetric  $n$ -tuple bid is a symmetric  $n$ -tuple bid (see Definition 3.9) which has an opening sequence of deltas satisfying the conditions of Theorem 15 and Corollary 3 and thereafter consists of consecutive sequences  $(\delta_i, \dots, \delta_j)$  from the following possibilities:*

- *Plain discounts so that the sequence is constrained as in Theorem 10.*
- *Decreasing discounts so that the sequence is constrained as in Theorem 12.*

Obviously we could also add in the possibility of quantity constraints here, but we have omitted it to keep things simple.

**Theorem 16** *Combinatorial auctions with general super/subadditive symmetric  $n$ -tuple bid are tractable and can be solved by nonbipartite matching.*

**Proof.** For the opening sequence the appropriate structure is a generalisation of Figure 10 (centre) to handle  $x$  objects. Thereafter the discounting sequences use the structures from Figure 9 (left and centre). If no object is allocated the structure still generates the revenue of the  $f_i$  edges, so this must be subtracted from the overall revenue. ■

This still leaves the asymmetric case open, with asymmetry we can define a triple bid where the surcharges are wired to particular objects. For example suppose we give a small surcharge between  $o_1$  and  $o_2$  and also between  $o_1$  and  $o_3$ , but if  $o_3$  is purchased after  $o_2$  we give a large surcharge. This would mean that the allocations  $\{o_1\}$  and  $\{o_1, o_2\}$  and  $\{o_1, o_2, o_3\}$  would have increasing values, something not possible in the symmetric case. The value of the triple  $\{o_1, o_2, o_3\}$  would exceed the values of the singletons it is composed of. However the loss of symmetry manifests itself if we allocate  $\{o_2, o_3\}$  as we then get a larger delta than we got for  $\{o_1, o_2\}$ . In this case the value of the triple does not exceed the value of the pair  $\{o_2, o_3\}$  and the singleton  $\{o_1\}$  (note that the  $\{o_2, o_3\}$  surcharge must give back the  $\{o_1, o_2\}$  surcharge in the allocation  $\{o_1, o_2, o_3\}$ ).

**Theorem 17** *Consider auctions which are solvable by graph matching techniques, and where each bid is placed on a bundle of objects, and consists of an XOR over valuations for every subset of that bundle (i.e. we are not considering auctions where indivisible bundles are present). In symmetric or asymmetric bids where the value of a complete set  $\mathcal{S}$  of objects is  $\mathcal{V}$ , there exists a partition  $\mathcal{P}$  of  $\mathcal{S}$  into subsets of size  $\leq 2$  such that  $\mathcal{V}$  does not exceed the sum of the values of these subsets.*

**Proof.** If there are no allocations which bring a greater increment than their singleton value, then let  $\mathcal{P}$  partition  $\mathcal{S}$  into singletons and we are done, because  $\mathcal{V}$  does not exceed the value of its singletons. If there are allocations which can bring a greater increment than their singleton value, then each such allocation must have been set-up by the allocation of some other object  $o_j$ . As stated in the proof of Theorem 15 (Part 1) the only way to set-up such an increase is when the end of one path removes an edge from the matching, so that a subsequent path can add an expensive edge which touches the removed end. A difference now with the symmetric case is that a number of subsequent allocations may get increasing value from this set-up. Furthermore, the object  $o_i$  which can get the maximum increase from the path which  $o_j$  has set-up (this means that  $o_i$ 's path has the most expensive edge touching the end of  $o_j$ 's path) may be able to get more value from a subsequent set-up path. We need some way to identify the appropriate pairs which will appear in the partition  $\mathcal{P}$ . Consider the maximum weight matching of the graph when the whole of  $\mathcal{S}$  is allocated. Now calculate the change in value  $\Delta_i$  resulting from removing each singleton  $o_i$  from the graph (and then replacing it before the next removal), and subtract the singleton value  $p_i$  from this giving  $\delta_i = \Delta_i - p_i$ . Also keep a record of the alternating path  $P_i$  which was applied to the graph when  $\text{mwm}(\mathcal{S})$  was transformed to  $\text{mwm}(\mathcal{S} - \{o_i\})$  (there is a slight abuse of notation here because  $\mathcal{S}$  is not a graph, but a set of objects; we mean the graph with all objects allocated). Now pick the  $o_i$  which has the greatest  $\delta_i$ , if its delta is positive then its path  $P_i$  must overlap with the last edge  $e_j$  at the end of another path  $P_j$ . The reason that they overlap rather than touch is that when the set-up path  $o_j$  was added its path did not include  $e_j$ , it merely touched one end of  $e_j$ . When  $P_i$  is subsequently applied it does include  $e_j$ . If  $o_j$  is subsequently removed from  $\text{mwm}(\mathcal{S})$  then  $P_j$  is applied and it does remove  $e_j$  from the matching. Now we know the pair  $\{o_i, o_j\}$  has a valuation which takes the maximal value from the set-up; we want this pair to be in our partition  $\mathcal{P}$ . We know that the increase brought by allocating the pair  $\{o_i, o_j\}$  alone is at least as large as the reduction in  $\mathcal{V}$  from removing them. This follows from Corollary 1 because the only way a pair could be worth more afterward than before is if the ends of their paths touched the ends of some existing paths, but since  $\{o_i, o_j\}$  touch each others' ends they can touch no others. Therefore when we move  $\{o_i, o_j\}$  out of  $\mathcal{S}$  and into our partition  $\mathcal{P}$ , we are putting at least as much value into the partition as we are taking from  $\mathcal{S}$ . We continue this process: identify the highest  $\delta_k$  in the remaining graph, find the  $o_l$  whose path overlaps  $P_k$ , move the pair  $\{o_k, o_l\}$  to the partition  $\mathcal{P}$ . When the highest remaining  $\delta$  is not positive we can take all the remaining objects as singletons and put them in  $\mathcal{P}$ . ■

Note that the claim in the theorem must say “does not exceed” instead of “equals” because the previous subsections on subadditive tuples have shown that it could be less. Also, we must say “there exists a partition” because it is not true that the value of the whole set cannot exceed the value of the subsets in *any* partition. Consider the quadruple, we have seen in Section 3.5 that the value of the whole quadruple could exceed the value of any two pairs within it. However in that case the value of the quadruple was less than the sum of its four singletons (hence the appropriate partition  $\mathcal{P}$  consists of four singletons). Alternatively we could construct a quadruple bid from two superadditive pairs, in which case the value of the quadruple would exceed the sum of its four singletons, and the appropriate partition  $\mathcal{P}$  would be the two pairs.



This Theorem is the most severe limitation we have encountered so far. It shows us that our  $n$ -tuple bids (even if asymmetric) are not true  $n$ -tuples, but collections of pairs and singletons. It is not possible for a bidder to express an increased valuation when there is a synergy among  $\geq 3$  objects, he can only express increased valuations for positive synergies among the pairs within the bundle. This is an important theorem as it shows the limit of our graph matching techniques when tackling superadditive auctions. It does not however prove that auctions which violate the condition are NP-hard, only that they cannot be solved by graph matching.

### 3.8.1 NOTE ON TENNENHOLTZ'S SUPERADDITIVE AUCTION

Tennenholtz (2002) has used the bipartite DCS structure depicted in Figure 11 to represent a triple bid where the first object is a forced sale, the second is allocated for more than the first (hence superadditivity) and the third is allocated for less than the second. Looking

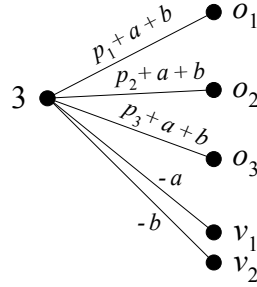


Figure 11: Tennenholtz's bipartite structure to model a superadditive triple bid. the vertex labelled 3 is degree constrained; it must have three matched edges incident.

at Figure 11 we can see that the minimum allocation is when one object is bought and  $v_1$  and  $v_2$  are matched; the revenue is then  $p_i$ , for whatever  $i$  is allocated. A constraint on the structure is  $a \leq b$ , therefore if two objects are allocated the  $-b$  link will be broken first; the revenue will then be  $p_i + p_j + a + 2b$ , so the increase in value ( $\delta = a + 2b$ ) is more than for a single object. If three objects are allocated the revenue will be  $p_1 + p_2 + p_3 + 3a + 3b$ , so the increase in value ( $\delta = 2a + b$ ) is less than for the second object. Recall that Theorem 2 said that no matter what clever bipartite structure we come up with, we cannot make a later addition to a graph generate a greater increase in value than an earlier one. How can a bipartite structure be used here to make the second allocation generate greater a greater increase in revenue than the first allocation? One can view the auctioneer as being forced to accept at least one object, before he makes any decisions about what to accept and reject. Therefore even though the allocation of the second object generates a greater increase than the first, the auctioneer could not choose to accept the second object in place of the first. Thus this does not violate Theorem 2 because the first allocation is no addition to the graph, it is the starting state.

The limitation of the structure is that the auctioneer could not accept any other higher bid for these objects; at least one object must be allocated to this bidder even if the price is poor. We can make an equivalent bid using our bipartite subadditive structure of Figure 5 (i.e. no need for DCS). Make the first allocation have a price  $p_i + 2N$  where  $N$  is the largest

weight in the graph, then subtract  $2N$  from the final revenue at the end. This guarantees that one of these first allocation links will be matched. Make the second allocation have price  $p_j + a + 2b$ , and the third have price  $p_k + 2a + b$ . Is it superadditive or subadditive? As we have formulated it it looks subadditive. As Tennenholtz (2002) has formulated it it could be seen as superadditive if one imagines the allocation of no objects, and then considers the increase in value after the first is allocated, and after the second is allocated.

## 4. The Space of Possible Bids

In this section we analyse the space of possible bids, to show what matching techniques are needed to handle each portion of the space, and to show which parts of the space are likely to be intractable. In between the regions that can be handled by graph matching, and the likely to be intractable (NP-hard) regions, there is an unknown region which can definitely not be handled by our graph matching techniques, but which we have not proved to be NP-hard.

### 4.1 Pair Space

To analyse the space of possible pair bids we need to consider the possibility that a bid could express an XOR over a different valuation for every possible allocation outcome. There are four possible allocation outcomes for a pair: (00), (01), (10), (11). Thus we could plot in a four dimensional space the regions in which bids are possible and what graph matching techniques are needed to solve them. We will pick a single 3D hyperplane in this 4D space to illustrate, that is the hyperplane where the valuation on (00) is always zero. This fixes the bidder's valuation for the empty allocation. As Myerson (1981) has shown, if an auctioneer wants to get maximum revenue from an auction, then payments for the empty allocation need to be considered; Myerson's work has shown that all possibilities can be necessary: an entry fee (a charge for getting nothing), payment to a bidder for getting an object, and payment to a bidder for getting nothing (in addition to the most well known case where a bidder pays for getting something). However this is for the pricing rule, which is a separate issue from the valuations the bidders express for each allocation. Still, we may want to consider the situation where a bidder expresses some valuation for getting the empty allocation, especially a negative valuation. This could be plausible in a scenario where the bidder will suffer some negative effect if he participates in the auction and gets nothing, as opposed to getting any object at all. This could be accommodated by adding  $v$  to the weights on the edges for the first allocation, and subtracting  $v$  from the final revenue for the whole auction. Then the final revenue for the auction is reduced by  $v$  iff that bidder gets nothing.

We do not depict this (00) dimension in Figure 12. The figure shows the 3D space (top left) and four vertical slices through the space (depicted below); on the top right is the diagonal plane through the space which shows the possible symmetric singleton bids (i.e. (01) = (10)). We do not need to consider values per unit greater than 1, as we can find the same bid by normalising to make the greatest of (01), (10), (11) have value 1. The boundary between "forced sale DCS" and non-forced sale (grey) occurs when the minimum singleton value is less than the pair value, as described in Section 3.2. The boundary between subadditive and superadditive occurs when the pair price exceeds the sum of the

singletons, and as noted by Theorems 2 to 4, it cannot be solved by bipartite matching, but as noted by Theorem 14 it can be solved by nonbipartite matching. Note that every point of the space can be solved by some graph matching technique.

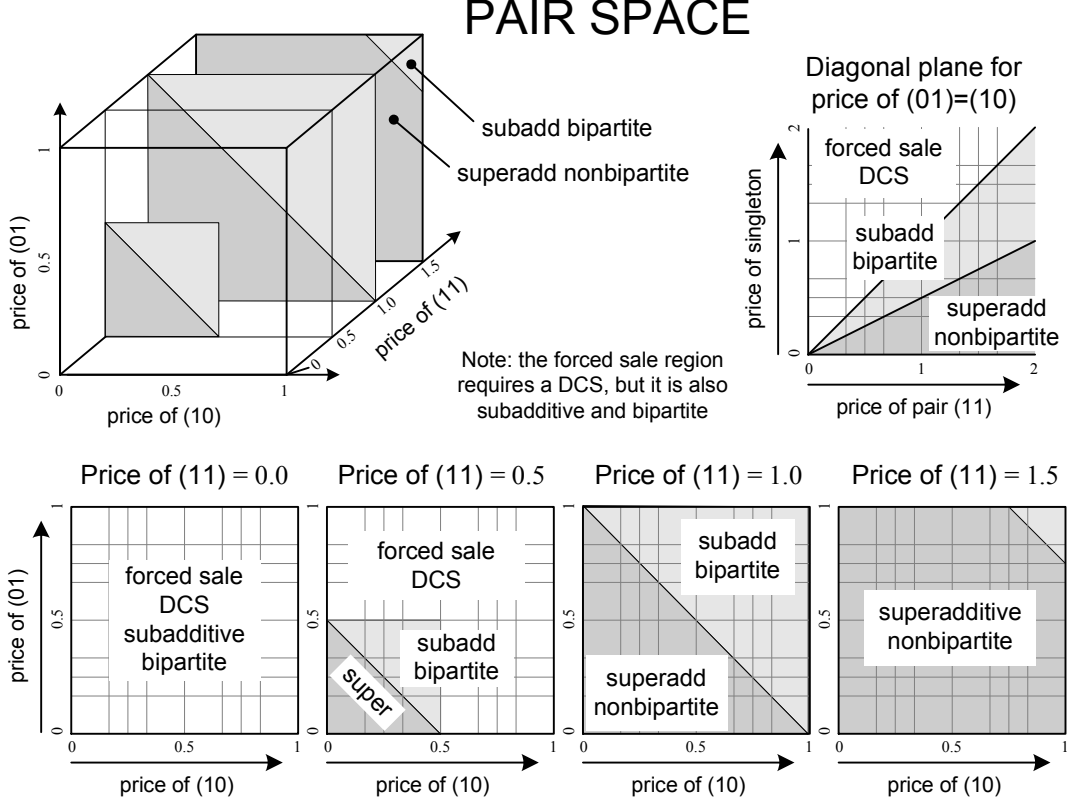


Figure 12: The regions of pair space in which bids are possible and the graph matching techniques needed to solve them.

**Theorem 18** *Combinatorial auctions which allow XOR bids over all subsets of a bundle, and where the maximum bundle size does not exceed two, are tractable for all possible bids, and the optimal allocation can be computed by solving a nonbipartite weighted DCS.*

**Proof.** In fact we do not need the full power of DCS in most cases, unless forced sales are required we can use regular weighted matching. For each subadditive pair bid which is placed, add the structure of Figure 3 to the graph. For each superadditive pair bid which is placed, add the structure of Figure 10 (left) to the graph. For each singleton bid placed simply add one vertex and connect to the object the bid is placed on, as described in the beginning of Section 3. If there is a bid where the discount for a pair is greater than the lowest singleton price offer in the pair, then we need to use a DCS; make the degree 1 ( $u = l = 1$ ) for each object on which such bids are placed. The maximum weight DCS in this graph gives the optimal allocation. ■

## 4.2 Triple Space

The space of all possible triple bids is an 8D space where a point represents an XOR over valuations placed on each possible allocation outcome: (000), (001), (010), (100), (011), (101), (110), (111). As with the pair case we do not depict the dimension where (000) varies. Furthermore we chunk together (001), (010), (100) and (011), (101), (110) into a single dimension each, implying symmetry among singletons and pairs. We just do this to give us a convenient number of dimensions to visualise; we have seen that complete asymmetry among singletons is possible, and limited asymmetry among pairs is possible (Theorem 13). This leaves us with a convenient three dimensions, representing the price offered for a singleton, pair, or triple; we will refer to these dimensions as  $s, p, t$ . The space is depicted in Figure 13. If we pick a 3D region within this space, we can use that to define constraints on allowed bids in an auction. If the region we pick is entirely within the areas indicated as solvable by graph matching techniques, then any auction which satisfies the corresponding bidding constraints will be solvable by graph matching. For example, consider the top face: the region indicated as “nonbipartite” has the total price of a pair and a singleton exceeding a triple (i.e.  $2p + s > 3t$ ), therefore it fits the constraints of Theorem 6, and an auction consisting of a collection of bids within this region can be solved by using the nonbipartite structure of Figure 8 for each bid.

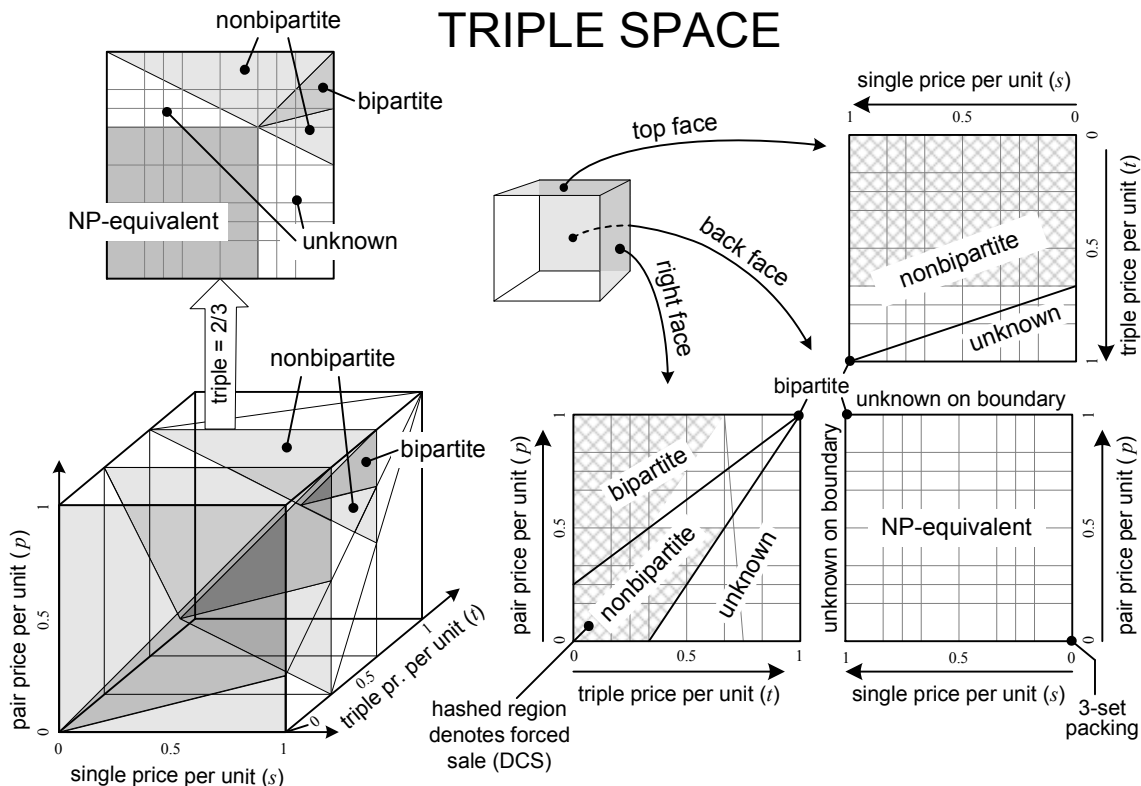


Figure 13: The regions of triple space in which bids result in tractable and likely intractable auctions. For the tractable regions we indicate the graph matching techniques that can solve them.

Note that we do not need to consider values per unit greater than 1, as we can find the same bid by normalising to make the greatest of  $\{s, p, t\}$  be 1. The graph has a lot of redundancy: if we consider any sub-cube contained within our unit cube (Figure 13, bottom left) and with a vertex at the origin, then the sub-cube is isomorphic to the unit cube. Alternately, if we trace a line from the origin, it will not cut through any regions (excepting that the origin point itself is a special case). For this reason we can represent the information in the unit cube by looking at these three faces: top face, right face and back face. The front face is almost identical to the pair graph for symmetric singleton bids (Figure 12, top right). The only difference is the nonbipartite region in the bottom right which corresponds to the triple case where  $d_2 \leq d_3 < 2d_2$  (see Section 3.4.2). The reason for this difference is that we are now not only concerned with a pair, but we also have to worry about forcing the triple price down to zero, and this can require a nonbipartite structure. Recall that  $d_2$  is the discount for a pair (the difference between a singleton and a pair), therefore  $d_2 = 2(s - p)$ ; similarly  $d_3 = 3(s - t)$ . Plugging these into the inequality gives us

$$2(s - p) \leq 3(s - t) < 4(s - p)$$

$$\frac{s + 2p}{3} \geq t > \frac{4p - s}{3} \quad (4.1)$$

This gives us the lower and upper bounds on this nonbipartite region. Only the bound  $t > \frac{4p-s}{3}$  is relevant for the front face from zero to one. The triple price on this front face is zero, so we have  $0 > \frac{4p-s}{3}$  or  $s > 4p$ , which gives us that bottom right nonbipartite region of the front face. We can calculate the region for the parallel planes behind the face similarly, by plugging in the  $t$  values  $\frac{1}{3}$  and  $\frac{2}{3}$ . The NP-equivalent region forms an oblique pyramid; the pyramid's base is the back face of the cube, and the pyramid's apex is just behind the cube's origin.

#### 4.2.1 RIGHT FACE OF UNIT CUBE

The nonbipartite region defined by Equation 4.1 appears in the centre of this face. On this face we have  $s = 1$  so Equation 4.1 becomes  $\frac{1+2p}{3} \geq t > \frac{4p-1}{3}$ ;  $t$ 's lower bound cuts the  $p$  axis at  $p = \frac{1}{4}$  and its upper bound cuts the  $t$  axis  $t = \frac{1}{3}$ ; the lines converge when everything equals one. The bipartite region is on the other side of the bound  $t > \frac{4p-1}{3}$ , which corresponds to  $d_3 \geq 2d_2$  (Section 3.4.1). The unknown region is on the other side of the bound  $\frac{1+2p}{3} \geq t$ , which corresponds to  $d_3 < d_2$ , as described in Section 3.4.3; this is superadditive, and as proved in Theorem 17, graph matching cannot handle it. We do not know if it is tractable. We also get the forced sale problem with triples, and we need to use a DCS. If the triple price per unit is less than one third of the singleton price we need to force a sale, and also if the triple price per unit is less than two thirds of the pair price. This gives us the hashed regions in Figure 12.

#### 4.2.2 TOP FACE OF UNIT CUBE

This face is mostly superadditive (and hence necessarily nonbipartite) as the pair price per unit exceeds the singleton. The boundary into the unknown occurs for  $3t > s + 2$ , i.e. the total triple price begins to exceed the total price of a pair and a singleton (recall that the

total price of a pair is fixed at 2 on this face); Theorem 17 has proved that graph matching cannot handle it, but again we do not know if it is tractable.

#### 4.2.3 BACK FACE OF UNIT CUBE (EXCLUDING UNKNOWN BOUNDARIES)

Firstly we show that any point on this face (excluding the unknown boundaries) is NP-hard. Let CAP3 denote the following problem: Finding the optimal allocation in a combinatorial auction which allows XOR bids over all subsets of a bundle, and with bundle size  $\leq 3$ . It is already known that the CAP3 is NP-hard. Rothkopf et al. (1998) showed that a special case of CAP3 is an instance of maximal 3-set packing<sup>3</sup>. This instance is the point indicated at the bottom right of the back face. This means that CAP3 is NP-hard because at least some subset of it is NP-hard, but as we have seen, CAP3 also has many subsets which are easy. It is important to know which subsets are easy and hard. We want to extend the region of known NP-hardness beyond this bottom right point, to prove that more subsets of CAP3 are NP-hard. Such a result is useful as it means that it would not be advisable to search for polynomial algorithms for those instances. When we look for a tractable subset of the CAP we are looking for some 3D region in triple space. We will show that any region which includes a point on this back face represents a subset of the CAP which is NP-equivalent.

Note that strictly speaking the term “NP-complete” is only applicable to decision problems, however it seems to be conventional in much of the AI literature to apply it to search problems (also called function problems); in this case what is really meant is that the search problem is both NP-hard and NP-easy, or in other words NP-equivalent. We will use the term NP-equivalent. To show NP-equivalence for a search problem we need to show that an NP-complete problem can be reduced to it, and that it has a polynomial reduction to an NP-complete problem.<sup>4</sup>

**Theorem 19** *Consider subsets of CAP3 which are defined by placing restrictions on the allowed bids; these restrictions take the form of constraints on the relative magnitudes of the prices offered for each of the possible subsets of a triple bid; the number of bids, or the objects they bid on are unconstrained. Let triple-greater denote a bid type where the price offer for a triple exceeds the sum of the price offers of the subsets in any partition of the triple. Let  $\mathcal{S}$  be the set of all subsets of CAP3 which allow at least some triple-greater bid type. Every element of  $\mathcal{S}$  is NP-equivalent.*

**Proof.** Any element of  $\mathcal{S}$  can solve any instance of Exact Cover by 3-Sets (X3C) and is hence NP-hard because X3C is NP-complete (Garey & Johnson, 1979, p. 53); to transform an X3C instance to any element of  $\mathcal{S}$  simply construct an auction where the  $m$  objects are the elements of the set from X3C (note  $m \equiv 0 \pmod{3}$ ), and the bids are all *triple-greater* bids with identical triple price  $t$ ; these bids are placed on the objects representing

3. General k-set packing was shown to be NP-complete by Karp (1972) by reduction from clique, and the special case of  $k \leq 3$  was shown by Garey and Johnson (1979) to be reducible from exact cover by 3-sets

4. This latter step seems not to have been done in the case of the CAP; a number of sources claim that the CAP is NP-complete (by which they mean NP-equivalent), but without proving this latter step (Rothkopf et al., 1998; Nisan, 2000; Sandholm, 2002). Sandholm (2002) claims it is NP-complete and cites Karp (1972), but the Karp paper only proves NP-completeness for the set packing decision problem. Note that de Vries and Vohra (2003) are careful to claim that the CAP is NP-hard while noting that the decision version is NP-complete.

the 3-element subsets from X3C. Note that all *triple-greater* bids have their triple price  $t$  exceeding the sum of any partition of the triple, therefore the value of an optimal allocation will equal  $tm/3$  iff there is an exact cover (otherwise the value of an optimal allocation must be less). The search problem of finding an optimal allocation is therefore at least as hard as the decision problem X3C (this proves the search problem is NP-hard).

To establish that it is NP-easy<sup>5</sup> we make a decision version of the problem. The input to the decision problem is an allocation value  $\mathcal{V}$  to be achieved and a set  $B$  of bids; each bid is a set of  $\leq 7$  bundles and price offers, at most one of these bundles can be accepted (i.e. XOR over the bundles in the bid). The decision problem returns “yes” if there exists an allocation which picks at most one bundle from each bid, and the total value is  $\geq \mathcal{V}$ . It is easy to see that the decision problem is in NP. Now we do a binary search (the same technique as used by Garey and Johnson (1979, p. 116)) to find the optimal allocation’s value  $\mathcal{V}^*$ . An upper bound on this can be found in linear time by computing the price per object of each price offer, finding the largest, and multiplying by the number of objects (similarly for lower bound). Having found  $\mathcal{V}^*$  we then go through the bundles one by one, deleting them from  $B$  and calling the decision problem again for  $(\mathcal{V}^*, B)$ . If the decision problem has a negative answer after any deletion, then we put that bundle back in  $B$ . Eventually  $B$  is the optimal allocation. It can be seen that the transformation is polynomial. ■

This result is stronger than showing that any region which includes a point on our back face represents a subset of the CAP which is NP-complete, because our back face only represents symmetric bids, whereas *triple-greater* bids are more general. To see that all the points on our back face represent *triple-greater* bids, note that they have  $t$  exceeding both  $s$  and  $p$ .

#### 4.2.4 UNKNOWN BOUNDARIES AND REGIONS

We now analyse the left and top boundaries of the back face and the adjacent unknown regions. We are not sure if there is a polynomial algorithm to solve auctions in these areas, but we can expect algorithms to perform much better here than in the NP-equivalent area. We cannot show that the left boundary is NP-hard as easily as the face, because if we give it a 3-set packing problem, it may come back with a solution which covers the object with singletons; similarly for the top boundary, it may come back with pairs. Nevertheless, if bidders’ price offers vary wildly, then it is unlikely to be optimal to accept a pair or singleton from a high bidder, so it would seem necessary to solve weighted 3-set packing. Therefore we can expect these areas (left and top boundaries) to be NP-hard. On the other hand, if price offers are comparable, a covering by pairs and singletons will give a reasonable allocation. For the unknown region on the right face: if a singleton is worth more than a triple, then even if there is an exact covering by 3-sets, we might prefer to replace an accepted triple by three singletons. This will happen when the differences between competing bids are small relative to the discounts. If there are a lot of bids relative to objects, then we can guarantee that an exact covering by singletons is possible. This happens if each object is covered by  $\geq 3$  bids (implying that the number of bids  $\geq m$ , the number of objects). To see this consider the bipartite graph  $(V_1 \cup V_2, E)$  formed by having one vertex in  $V_1$  for every bid, and all the objects in  $V_2$ , so that each bid in  $V_1$  connects to three objects in  $V_2$ .

---

5. This solution was proposed to me by Vincent Conitzer.

Now any subset of the objects must be covered by a set of bids which is at least as large, it is impossible for a small set of bids to cover a larger set of objects to a depth of three; this guarantees that each object can be matched (see Hall's theorem, (Bondy & Murty, 1976, p. 72)). Similarly for the unknown region on the top face: if a pair is worth more than a triple, we might prefer to replace two accepted triples by three pairs, which will happen when the differences between competing bids are small relative to the discounts. Guaranteeing that an allocation consisting entirely of pairs exists is more difficult than the analogous condition in the case of accepting singletons. This is because for pairs we are matching two objects, which cannot be matched by representing a bid with a single vertex, as we did in the bipartite graph for the singleton case. However we could summarise by saying that, for both of these regions, if objects are covered by many bids, then ignoring triples and calculating a matching of pairs and singletons is likely to lead to a close to optimal allocation. On the other hand if bidding is sparse, calculating the exact optimal allocation may well be feasible; exact cover by 3-sets is solvable in polynomial time if no element is covered by more than two subsets (Garey & Johnson, 1979, p. 221). Finally if the bidding is mixed, with some objects heavily covered, and others sparsely, then the problem may well be intractable.

Indeed the above arguments also apply across the NP-equivalent area of the back face: as we get closer to the unknown boundaries, algorithms can be expected to have a better performance; this would apply to approximation algorithms in particular, e.g. if  $s \approx 1$  and each object is covered by  $\geq 3$  bids and competing bids are similar, then a covering by singletons will be close to optimal (and this is easy to compute by treating bids as quantity constrained with  $q = 1$ , see Section 3.1).

### 4.3 Quadruple Space (and beyond)

We will describe the quadruple case briefly as the region boundaries are calculated in a similar fashion as for the triples. This time we fix  $s = 1$  and we look at  $p$  versus  $t$  for various values of  $q$  (the price per unit of a quadruple). The left diagram of Figure 14 shows a view of triple space using this same perspective, for comparison purposes; it is interesting to see how close matching techniques can take us to the region of known NP-equivalence.

The diagram to the right of this (Figure 14 for  $q = 0$ ) has a more limited tractable region because of the requirement that  $q = 0$ ; if  $p$  is quite small, then discounts on  $t$  and  $q$  need to be greater, making it difficult to achieve  $q = 0$ . The three labels  $a, b, c$  indicate the structure needed to solve auctions in the region labelled. The structures are the three subadditive triple structures of Section 3.4. Label  $a$  is for the bipartite structure of section 3.5.1 with constraints  $d_4 \geq 2d_3 - d_2$  and  $d_3 \geq 2d_2$ . Label  $b$  is for the nonbipartite structure of section 3.5.2 with constraints  $d_3 \geq d_2$  and  $d_4 \geq d_3 + d_2$ . Label  $c$  is for the nonbipartite structure of section 3.5.3 with constraints  $d_4 \geq d_3 > 2d_2$ . As in the triple diagram, we can see how the bipartite is just a special case within the nonbipartite (general matching) region.

The diagram to the right of this (Figure 14 for  $q = 1$ ) shows the square of NP-equivalence for 4-set packing at the origin; this square grows larger and larger as  $q$  increases. Note that we need the full power of our superadditive structures to achieve the tractable region in this  $q = 1$  diagram. We use a structure similar to Figure 10 (right), except removing the



# TRIPLE SPACE

# QUADRUPLE SPACE

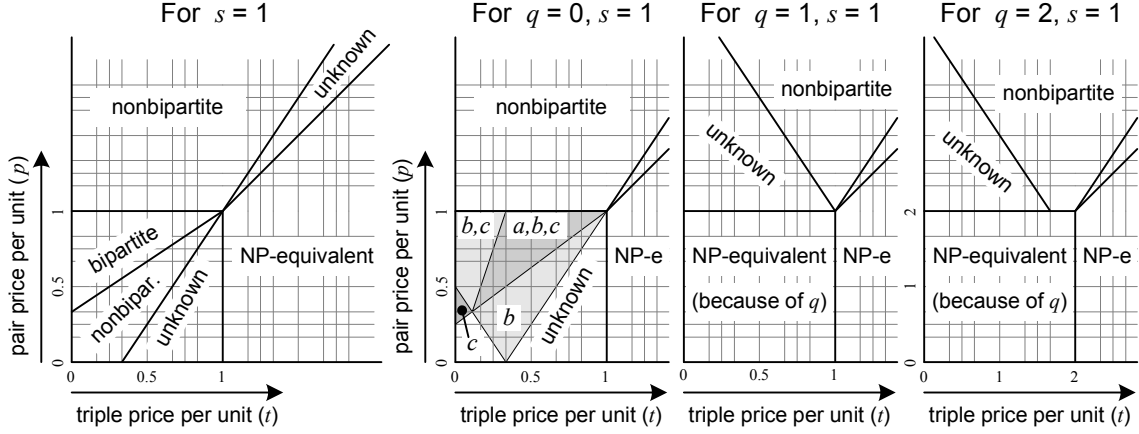


Figure 14: The left diagram shows an alternative view of triple space, for comparison. The remaining diagrams show the regions of quadruple space in which bids result in tractable and likely intractable auctions. For the tractable regions we indicate the graph matching techniques that can solve them.

vertex with the  $f_2$  edges. With this we can achieve any pair surcharge, and any subsequent discount for the triple and quadruple. The diagram also shows a limit encroaching on the superadditive region (the line coming from the top left). This is because with  $q = 1$  the price of a quadruple is 4, and if  $t$  is small we need a substantial second surcharge to make the jump from  $3t$  to  $4q$ ; when  $p$  is also small the first surcharge is small, and therefore a large second surcharge is unachievable. What we are seeing here is the constraint due to Part 1 of Corollary 2, i.e.  $inc_i \leq inc_{i-2}$ . In this diagram we have fixed  $s = 1$  and  $q = 1$ ; our constraint is that the second surcharge cannot exceed the first; this means  $2p - s \geq 4q - 3t$ , which gives us  $p \geq \frac{5-3t}{2}$ .

The diagram to the right of this (Figure 14 for  $q = 2$ ) also shows this limit, but we see that it is covering relatively less area. As  $q$  increases its slope remains constant, and its intercept with the square of NP-equivalence happens at  $t = \frac{2q+1}{3}$ , which is  $1\frac{2}{3}$  when  $q = 2$ . Note that we are getting much more reach here than in the analogous triple case, because the quadruple price can exceed the triple price plus the singleton price ( $4q > 3t + s$ ), whereas the triple price cannot exceed the pair price plus the singleton price ( $3t \not> 2p + s$ ). To see this in the diagrams notice that if we increase  $t$  in the leftmost diagram (Figure 14) we are forced to also increase  $p$ , or we quickly get into NP-equivalent territory; in contrast, in the rightmost diagram (Figure 14 for  $q = 2$ ) if we imagine increasing  $q$  (moving in the third dimension), we see that it is the price of  $p$  relative to  $q$  that is the more limiting factor, rather than the price of  $t$  relative to  $q$ .

Imagining the space of  $n$ -tuple bids we can see that the square of NP-equivalence in Figure 14 (right) will appear in every new dimension we extend to: offers where the  $n$ -tuple price exceeds its partitions will always be NP-hard for  $n \geq 3$ . This is the major limitation on our auctions; for large  $n$  the vast majority of the space is NP-hard. However

when we move to the region which has superadditivity for pairs, this will allow a certain amount of superadditivity in the quadruple, sextuple, octuple, etc. so we will have some tractability along this dimension. Note that by “superadditivity” here we mean specifically that the increase in value due to adding another object can exceed the singleton value of that object. If we broaden “superadditivity” to mean that the  $n$ -tuple exceeds the sum of all its singletons, then we also have some tractability along the dimensions for the triple, quintuple, septuple, etc. As for the subadditive case we will see, as in Figure 14, that this also gets less coverage as we increase  $n$ . However, most of the subadditive regions which cannot be covered by matching techniques correspond to auctions where subsets of the  $n$ -tuple have large discounts, but the whole  $n$ -tuple has a small discount relative to these subsets. This has a superadditive flavour (later additions bring a greater increase in revenue than earlier ones). This is probably a rather unlikely bidding region, it may make sense for the bidders to simply bid on smaller tuples (the large discounts on the larger sets effectively mean those sets are worthless).

#### 4.4 Discussion of Limitations

Here we will discuss what can and cannot be done, and why. We will briefly discuss three issues:

1. Why is subadditive easy with graph matching, but superadditive is not?
2. Why is a superadditive pair solvable with graph matching, but a triple is not, yet subadditivity suffers no such limit in moving from two to three or beyond?
3. Why is two easy, but three is NP-hard?

For the first question we need to go back to Corollary 1. Our graph must allow singletons to be purchased at unit price. Therefore no single object can release any more value from the graph than its singleton value *unless* a previous allocation has set it up for an increase (by ends of paths touching). As we have seen, these set-ups are limited to pairs. What we really want for superadditivity is that later additions could cause larger and larger increases in value. Maximum weight graph matching is biased towards maximising; if there is more value to be extracted, it will extract it immediately. This is no problem for subadditivity, because the best links are taken first by the matching algorithm, leaving lower cost ones for later additions. We could reverse this bias and use minimum weight graph matching instead; however we would then have the problem that the auctioneer would now be trying to take the lowest bids offered for each object. What we really want is maximising on the vertices representing objects (i.e. to pick the maximum weight incident edges, from all those edges offered by different bids), but minimising on the vertices representing bids (for each bid structure it should match the edges offering lowest weight first; this would require a maximum cardinality, minimum weight matching). We could not expect to find a polynomial algorithm for this matching problem however. Indeed having a solely maximising bias is what makes it easy to make a polynomial algorithm; there is one direction to the algorithm: always apply alternating paths which increase the weight of the matching.

For the second question we are looking at the fundamental limitation of 2D-matching. An edge in a 2D graph connects only two things, 3D matching in hypergraphs could get

over this, but polynomial algorithms are not known. In the 2D case, because an edge can connect two things it can express relationships over two things, and hence we can have a superadditive pair by making a special edge which is triggered if two objects are allocated. We can also turn off an edge if two objects are allocated, or we can handle situations where one object is allocated and the other is not. Hence we can fully express any relationship among two things. This is not true for three or more. In the subadditive case we do not have full expressiveness for three, four and beyond (i.e. subadditivity does suffer a limitation in moving from two to three and beyond). We have seen in the asymmetric subadditive case (Section 3.7) that out of the  $\binom{n}{k}$  possible subsets of size  $k$  a maximum of only  $n - k + 1$  can be given a discount. We cannot express relationships among  $n$  objects, we can only count how many have been allocated and force a different edge to be taken when the number of objects already allocated has exhausted all more expensive paths.

For the third question we want to see if graph matching can give us any insight into why problems like set packing are easy for two but hard for three. We can simplify things by ignoring weights. Consider auctions where bids are placed for “all or nothing”<sup>6</sup> bundles and all prices are one.<sup>7</sup> We know that this auction is tractable for bundles of size two (pairs), but NP-hard for bundles of size three. To solve the pairs case by graph matching we can use Rothkopf et al.’s (1998) structure where the vertices consist solely of the objects, and the bids are edges connecting pairs of objects. To have a polynomial algorithm we need some way to search the space of solutions *incrementally*, making some decisions on which we will never have to backtrack; whereas a brute force algorithm would not make any definite decision before completing an exhaustive search. We can see how graph matching does this in the pairs case. Suppose we start with some greedy allocation that simply matches every edge it finds which has both ends exposed. We have already found some definite information: every vertex matched by this greedy method will be matched in the final allocation. The graph matching algorithm works by successive augmenting paths, which will never expose a matched vertex. From the point of view of the auction it means that once we match two objects  $o_1$  and  $o_2$  in a bid, we are certain that both of them will sell.

Now let us consider all or nothing bids for triples, with price one. This corresponds to matching in a 3D hypergraph. If we do a greedy allocation, and match three objects, there is no guarantee that any of them will sell (i.e. some will sell, but one cannot identify which ones after the greedy phase). Say the graph has seven objects and three (hyper)edges, so that matching six objects is possible, using just two of the edges. Also assume that all objects are incident to some edge. We may have picked the wrong edge in our greedy phase, i.e. the edge incident to the seventh object. Consider the analogous situation in the 2D case: the graph has five objects and three edges (and all objects are incident to some edge), so that matching four objects is possible, using just two of the edges. In this case, whatever edge we initially pick will be part of the maximum matching because there are two optimal solutions. This is one factor that makes the 2D case easy: it has more optimal solutions, whereas the optimal 3d matching is more likely to be unique. This however is not the crucial difference. Consider the 2D case again, but this time suppose one of the five objects is not connected with the rest of the graph. Now one of the edges in the graph is

6. Nisan (2000) calls these “single minded” bids.

7. Note that  $b$ -matching cannot handle all or nothing bids. If we connect a vertex with a  $b$ -value  $n$  to an  $n$ -tuple, it must match all, and nothing is not an option.

adjacent to the remaining two, and it does not appear in the maximum matching. Suppose we picked that edge greedily; it is still true that both objects matched by that edge will appear in the maximum matching, and we can achieve the maximum matching by a single augmenting path which removes this edge and adds the two edges touching its ends. Thus a partial matching in the 2D case can be transformed into an alternative one by following an augmenting path, and in following the path from one end to the other, each flipped edge will force *one* adjacent edge to be flipped. In the 3D case if we want to transform a partial matching, each new edge we try to add may force us to change *two* adjacent edges, and changing these adjacent edges may force us to change four, eight, sixteen ...; so in the worst case we have an exponentially growing tree of changes; this tree is the analogue of an alternating path in the 2D case.

## 5. Complexity Results

This section gives the time complexity for computing the optimal allocation in the auctions described in Section 3. We group the auctions in two classes: those solvable by bipartite matching and those solvable by nonbipartite matching. Finally, in Section 5.3, we look at how some tricks can be employed to handle Quantity Constrained and Multi-Unit bids.

### 5.1 Optimal Winner Determination in Auctions with Bipartite Structures

**Theorem 20** *For combinatorial auctions with  $m$  objects for sale, allowing OR-of-XOR bids, where each XOR is a quantity constrained discounted subadditive symmetric  $n$ -tuple bid as described in Theorem 11, and where the maximum size of a tuple is  $s$ , and there are  $b$  tuple bids, and  $N$  is the maximum value of a singleton price offer, the optimal allocation can be computed in  $O(\sqrt{m+bs} bs^2 \log(N(m+bs)))$  time. The speed up factor of Section 2 can improve this by a factor of  $(2 - \frac{\log bs^2}{\log(m+bs)})$ .*

**Proof.** We use bipartite matching, using Gabow's scaling algorithm (see Section 2). The maximum number of vertices is  $m+bs$ , i.e.  $m$  objects and a maximum of  $s$  bidding vertices for each of the  $b$  tuple bids. The maximum number of edges is  $bs^2$ , i.e. no more than  $s^2$  edges for each of the  $b$  tuple bids (the number of edges is actually  $\frac{n(n+1)}{2}$  for normal discounted subadditive symmetric  $n$ -tuple bids). ■

From this we can see that the performance scales much better with increasing numbers of bids than with increasing tuple size. The number of objects hardly influences the performance. Note that the construction of the graph, from the tuple bids, takes time which is linear in the size of the graph (and hence insignificant compared to the matching algorithm's time complexity), but polynomial in the size of the tuples. The space requirement is likewise roughly  $bs^2$ .

### 5.2 Optimal Winner Determination in Auctions with Nonbipartite Structures

**Theorem 21** *For combinatorial auctions with  $m$  objects for sale, allowing OR-of-XOR bids, where each XOR is a general super/subadditive symmetric  $n$ -tuple bid as described in Definition 3.10 (or indeed any other bid for which we have provided a graphical structure), and where the maximum size of a tuple is  $s$ , and there are  $b$  such tuples, and  $N$  is the max-*

*imum weight of a singleton price offer, or a superadditive increment (whichever is greater), the optimal allocation can be computed in*

*$O(\sqrt{(m+bs)} \alpha(bs^2, m+bs) \log(m+bs) bs^2 \log(N(m+bs)))$  time, which is roughly  $O(\sqrt{(m+bs)} \log(m+bs) bs^2 \log(N(m+bs)))$ .*

**Proof.** We use nonbipartite matching, using Gabow’s scaling algorithm (see Section 2). The maximum number of vertices is  $O(m+bs)$ , i.e.  $m$  objects and a maximum of  $2s$  bidding vertices for each of the  $b$  tuple bids. The maximum number of edges is  $O(bs^2)$ , i.e. a maximum of  $s^2 + s - 2$  edges for each of the  $b$  tuple bids. ■

The expression looks long, but saying it is roughly  $O(bs^2)$  would not be far off. The same comment as above (in Section 5.1) on performance and graph construction applies here.

### 5.3 Efficient Handling of Quantity Constrained and Multi-Unit Bids

In a quantity constrained bid all the vertices from the bid (left, or  $V_1$ ) side are identical (we represented it with  $\frac{n(n+1)}{2}$  edges in Figure 1, but it is easy to see how the bidding vertices can be made identical, using a total of  $n^2$  edges). Identical vertices can be compressed before the weighted matching algorithm is called, and expanded after. The same applies if multiple units are on offer. Suppose that each bidder only desires a single unit of an object, but many bidders want a unit of this particular object. The uncompressed representation would have a vertex in  $V_2$  for each unit of the object, and each bidder would place a QCMO bid (with  $q = 1$ ) covering each unit. In effect the bidders are placing an XOR over all the units, to maximise their chance of getting one. All of the units of the object now have identical incident edges. In the compressed version we can represent that object by a single vertex, and expand it out to multiple identical vertices later. If bidders desire a number  $g$  of units ( $g > 1$ ), but do not decrease their price for larger quantities, then in the uncompressed form they can place a QCMO bid (with  $q = g$ ), which covers all the identical units. Now the  $q$  vertices in  $V_1$  of the QCMO bid are identical, in addition to the object vertices in  $V_2$ . In the compressed form such a QCMO bid would have one vertex each in  $V_1$  and  $V_2$ . These vertices will need to be expanded out later along with some of their edges, but many of the edges will be lost after the compression/expansion procedure. If bidders bid subadditively (or superadditively, although this is unlikely for identical units) over the multiple units (e.g. buy more, pay less per unit), then we need to expand out the individual units in  $V_2$ ; we can represent each unit by a vertex, ordered vertically on the right, and for each bid add its edges starting at the top and moving downward. When all bids are added in this way, any identical vertices can be compressed; most units towards the bottom should have identical edge connections (representing the bids that did not differentiate their price over multiple units).

To understand the compression: consider a graph and its minimum weight cover  $C$ , if we take any vertex  $v$  in a graph  $G$  (with cover  $C(v)$ ) and duplicate it (and all of its incident edges), then the minimum weight cover of the new graph including the duplicate must assign an identical cover  $C(v)$  to the new vertex and will otherwise be identical. The compression procedure computes the minimum weight cover (the matching algorithms compute the cover anyway) of the compressed graph, and then expands all compressed units to their original

number of vertices. Note that any edges which are not in any maximum weight matching<sup>8</sup> compressed graph can be deleted, as they will not appear in the maximum weight matching of the uncompressed graph either. Since we have computed the cover, this means we delete any edge  $e = \{u, v\}$  for which  $C(u) + C(v) > w(e)$ . What is happening here is that in the compressed graph the algorithm can already decide which edges are definitely bad and will not appear in the final matching of the uncompressed graph. Finally the maximum weight matching of the uncompressed graph is *recovered* by transforming the cover to a matching in  $O(\sqrt{V}E)$  time. This is a simple matter of duplicating the graph and connecting uncovered vertices by zero weight edges; a maximum cardinality matching is then calculated on this graph (see Procedure Recover-Max-Matching in Kao et al. (2001); their paper works with bipartite graphs, but this particular procedure is also correct in general). Intuitively one can view the computation with the compressed graph as first deciding which bids are good and might be accepted (positive cover on their vertices) and deciding which bids should definitely be rejected (zero cover on their vertices); in its second (recover) phase it decides which of these good bids should actually be accepted, based on the number of units available.

To get an idea of the efficiency this brings consider the bipartite auction of Theorem 20 which has time complexity  $O(\sqrt{m + bs} \, bs^2 \log(N(m + bs)))$ . We will just focus on the multiple units issue. Suppose that the  $m$  objects consist of  $j$  types of object, with  $k$  units of each ( $m = jk$ ). Further suppose that bidders do not decrease their price offers if buying multiple units and that each bid is for  $g$  units of an object. Thus we have  $jk$  vertices in  $V_2$  and  $bg$  vertices in  $V_1$ . Each of the  $bg$  vertices in  $V_1$  has  $k$  edges (one to each unit of the object it is interested in), giving a total of  $bgk$  edges. We have uncompressed time complexity  $O(\sqrt{jk + bg} \, bgk \log(N(jk + bg)))$ . The compressed graph has  $j + b$  vertices and  $b$  edges. The total complexity for calculating the minimum weight cover of the compressed graph is  $O(\sqrt{j + b} \, b \log(N(j + b)))$ . after this we delete edges  $e = \{u, v\}$  for which  $C(u) + C(v) > w(e)$ , in linear time. Now when we expand this graph we get back a maximum of  $bgk$  edges (we may lose some, but this depends on edge weights and we will not go into this level of detail). The complexity for recovering the maximum weight matching will be  $O(\sqrt{jk + bg} \, bgk)$ , and this dominates the time taken to calculate the cover. In total then we have lost the log term when we use the unweighted matching algorithm for the uncompressed graph. The  $bgk$  term is still the dominant part. Note that  $bs^2$  in Theorem 20 corresponds to  $bgk$  in our example because  $s^2$  was just an upper bound for the more precise  $gk$  in our example. Finally, we note that the  $s^2$  in the original complexity is a bottleneck that will not go away, especially if bidders place large  $n$ -tuple bids with different prices across objects; the computation of the optimal allocation will take quadratic time, even if some of those objects are identical.

Tennenholtz (2000) describes a dynamic programming approach to the multi-unit auction problem, however it requires a rather large graph to be constructed (its size is exponential in the number of types of object for sale). Tennenholtz (2002) suggests using  $b$ -matching, making the object vertices have a  $b$ -value corresponding to the number of units of that object for sale. This is not as efficient as the above approach, as the weighted matching algorithm itself would have to make the decisions about how many to allocate, rather than just deciding which is a good bid (as above).

---

8. A maximum weight matching is not necessarily unique; if there are many possible matchings, then we can only delete those edges which do not appear in any.

## 6. Related Work

Rothkopf et al. (1998) seems to have been the first work to relate the CAP to graph matching; however it seems that Tennenholtz’s idea to use  $b$ -matching was independently developed because he only mentions Rothkopf et al. briefly and does not discuss their matching approach. Rothkopf et al.’s matching approach is rather different to Tennenholtz’s (and ours). They consider the case where bundles are of size  $\leq 2$ . Their graph does not represent all the bids of bidders, but only the maximal bid on each bundle. Therefore there is a preselection where all bids which are dominated (by a bid on the same bundle) are discarded. Bidders are not represented in their graph, but objects are. There is an edge between two objects if a binary bid has been placed on them (its weight will be the maximum bid on that bundle). For singleton bids there is one extra vertex per object, with an edge between, to represent the maximal singleton bid on that object. This formulation means that a bidder can be allocated all the objects he bids on; it is not possible for a bidder to express a quantity restriction on how many objects he is willing to buy. Another limitation is that a bidder cannot place a subadditive bid; i.e. a bidder cannot express “I will pay  $p_1$  for this and  $p_2$  for that, but only  $p_1 + p_2 - disc$  for both”; the auctioneer would simply match the singleton bids because they are worth more. In other words Rothkopf et al.’s language only allows OR, not XOR. Rothkopf et al. give an  $O(m^3)$  bound for solving the problem, where  $m$  is the number of objects; this assumes that a bid has been placed on every possible pair and every singleton. This time bound can actually be improved to  $O(\sqrt{\alpha(m^2, m)} \log m \, m^{2.5} \log(mN))$  using Gabow and Tarjan’s (1991) algorithm. To see this use the graph matching bound  $O(\sqrt{V\alpha(E, V)} \log V \, E \log(VN))$  and note that  $|V| = 2n$  and  $|E| = m^2 + m$ . In practice placing a bid on every possible pair is unrealistic in large auctions, because there are  $m(m+1)/2 \approx m^2/2$  pairs; if only  $n$  bids are placed on pairs we have instead the bound  $O(\sqrt{m\alpha(n, m)} \log m \, n \log(mN))$  which takes into account the possible sparseness of the bids. To compare with our approach, we can model the same scenario with our superadditive bids on binary bundles (see Figure 10), hence our graph will have  $2n$  edges more than Rothkopf et al.’s, but is otherwise identical.

Rothkopf et al. (1998) also describe a number of other special cases of the CAP which are tractable, one of these is the case of linear goods: this means the goods are ordered, and bids can only be placed on a contiguous sequence; this makes sense when bidding for parts of a coastline (for example), or for some time interval during which a resource can be used. Rothkopf et al. (1998) however show that extending from goods ordered along one dimension to rectangles in 2D space makes the problem NP-hard. Tennenholtz (2002) has tackled the linear goods case with graph matching; we did not address this as we have nothing to add.

Nisan (2000) has a quite comprehensive work on the relation between linear programming (LP) and the CAP. By looking at the relaxation of the integer programming formulation of the CAP, Nisan (2000) notes how the solution would be straightforward if objects could be divided, so that a bidder could get  $2/3$  of an object for example (which could make sense for raw materials, like oil, or for bandwidth). Interestingly, Nisan concludes that “the main difficulty does not come from the effects of substitutabilities or complementarities ... the difficulty lies in the indivisibility of the goods which is what can not be handled well”. In contrast from our graph matching perspective our conclusion is that the difficulty

comes very much from the complementarities (superadditivity). However when we say that matching techniques can handle substitutabilities (subadditivity) we must remember that we are not talking about the full CAP; we have seen that we can really only handle the special case of symmetric subadditivity.

Nisan (2000) also identifies classes of auctions where the LP solution will give integer results. This covers some of Tennenholtz's (2000) classes, such as linear goods, and some of Rothkopf et al.'s (1998) classes, such as hierarchical bids (which Rothkopf et al. call nested structures). Also included are: an OR-of-XORs of singleton bids, which corresponds to QCMO bids (see Section 3.1) when  $q = 1$ ; downward sloping symmetric<sup>9</sup> bids, which are a special case of our subadditive symmetric  $n$ -tuple bids (see Theorem 10), where singleton prices are identical; auctions over a set of objects  $S$  which are the sum of two auctions (which have integral solutions) over disjoint subsets  $Q$  and  $R$  ( $Q \cup R = S$ ).

More results in this area are given by de Vries and Vohra (2003); their work gives a good overview of methods used to tackle the CAP; these include tractable instances, exact but exponential methods, approximate methods, market mechanisms, and more. They note that the typical formulations of the CAP (allowing XORs via dummy goods, or directly encoding XORs) are an instance of the set packing problem (SPP), a well studied problem (Balas & Padberg, 1976)<sup>10</sup>. In terms of identifying solvable instances of the SPP, they cover much of the same ground as Rothkopf et al. (1998), and much more, but from an integer linear programming perspective. De Vries and Vohra (2003) formulate the SPP as follows.  $M$  is the set of objects for sale.  $V$  is the set of subsets for which prices have been offered.  $A$  is a matrix where each row corresponds to an object and each column corresponds to a member of  $V$ . The entries in  $A$  are  $a_{i,j}$  ( $i$  = row,  $j$  = column) where  $a_{i,j} = 1$  if object  $i \in M$  appears in subset  $j \in V$ , and  $a_{i,j} = 0$  otherwise. The optimisation problem is to find a vector  $x$  of ones and zeros, which selects members of  $V$  or rejects them (select=one, reject=zero), and which maximises the total value of the selected members, subject to the constraint that no object appears in more than one member (recall that each member of  $V$  is a subset of the objects). De Vries and Vohra (2003) then identify constraints on the  $A$  matrix which guarantee that the SPP has an interpretation as an easy graph matching problem. The first case identified is when each column of  $A$  has at most two ones. This then corresponds to the graph matching case identified by Rothkopf et al. (1998), discussed above. De Vries and Vohra (2003) state that "Instances of SPP where each column has at most  $K \geq 3$  non-zero entries are NP-hard" this is a little ambiguous, but it is clear that they mean "*some* instances of SPP...", because in restricting each column to two ones they are aiming to provide a condition which is sufficient to guarantee tractability, but not necessary. We have seen that we can handle many instances which do have multiple ones

---

9. Note their definition of symmetric means that the valuation on a subset relates only to its size, and not the singleton valuations on the objects.

10. Balas and Padberg (1976) formulate set packing as a special case of set partitioning. We must be wary of terminological confusion here again: for the set packing and set partitioning problems, Balas and Padberg (1976) use the acronyms SP and SPP respectively, while de Vries and Vohra (2003) use SPP and SPA respectively.



per column. For example, the triple bid of Figure 6 corresponds to the following  $A$  matrix:

$$\begin{array}{c} \begin{array}{cccccccc} & \{o_1, o_2, o_3\} & \{o_1, o_2\} & \{o_2, o_3\} & \{o_1, o_3\} & \{o_1\} & \{o_2\} & \{o_3\} \\ \begin{array}{l} o_1 \\ o_2 \\ o_3 \end{array} & \left[ \begin{array}{cccccccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array} \right] \end{array}$$

De Vries and Vohra (2003) also state that “Instances of SPP with at most two non-zero entries per row of  $A$  are NP-hard”, which we would again refine with *some*.

Sandholm (2002) also outlines a number of special cases which admit polynomial solutions, but these come from the related work which we have described above. None of the above related works cover our generic  $n$ -tuples; it would appear that our  $n$ -tuples are a class of tractable auctions that has not yet been identified. In terms of the SPP’s  $A$  and  $V$ , we have relaxed the conditions which are sufficient to guarantee tractability. We cannot come up with any condition on  $A$  which is necessary and sufficient to ensure a graph matching solution, because our constraints also relate to the values in  $V$ .

Another research area, within combinatorial auctions in AI, involves algorithms which find exact solutions for the full CAP. Such algorithms are obviously exponential in the worst case (since the problem is NP-hard), but clever algorithms can solve many common instances quickly. For example Sandholm, Sandholm and Suri (2002, 2003) use a great range of tricks to find an optimal allocation quickly. One of the ideas of Sandholm (2002) is to capitalise on the sparseness of bids (unlike dynamic programming approaches), which happens automatically with our matching approach; sparse bidding results in a graph with a small number of edges  $E$ , and  $E$  is by far the greatest contributor to the computational complexity. A further trick employed by Sandholm and Suri (2003) is to recognise tractable special cases (including many of those described above) and exploit them. These cases could arise at subnodes during the search for an optimal allocation. This is a possible role for the tractable instances we have identified; they could be incorporated in a general CAP solver. With this it would be possible to handle all or nothing bids; the search algorithm could run graph matching for a subset of the objects, and compare the result with what all or nothing bids are offering for that bundle. One attraction of using graph matching algorithms is that these algorithms are extremely efficient, given that they have been improved on several times over the last fifty years; therefore if we can use these to solve some special cases, we know that we are using some of the most highly developed techniques for solving tractable problems.

To gain advantage by incorporating graph matching techniques in a general solver would require techniques that can quickly identify, from a set of bids, if they are solvable by graph matching. Clearly this is easy if bids are submitted in the form of our  $(o_1, p_1, \dots, o_n, p_n, \delta_2, \dots, \delta_n)$  tuple representation (we simply scan through the deltas checking our constraints), but if bids are submitted in other languages it seems that our cases could require exponentially long bids. Nisan (2000) discusses bidding languages, such as XOR-of-ORs, OR-of-XORs and OR\*. As we stated previously, our  $n$ -tuples represent an XOR over their subsets, and a number of such bids means an OR-of-XORs. However, no bidding language will be as concise as our  $(o_1, p_1, \dots, o_n, p_n, \delta_2, \dots, \delta_n)$  tuple representation, and the above languages will in general need an exponential expansion to represent our  $n$ -tuples. This is because our tuples allow unique prices for singletons, and hence it is most

likely that each possible subset of size  $k$  in the bid will have a unique price (this happens when the sum of each subset is unique). In contrast, Nisan’s downward sloping symmetric bids have an identical value for each subset of size  $k$ ; this is what allows him to represent it in the OR-of-XOR language in size  $n^2$ . The above languages are unable to exploit the redundancy due to symmetry in our  $n$ -tuple bids. On the other hand we could consider our  $n$ -tuples as a new bidding language, which gives a concise representation for our bid types; a general purpose solver might be configured to accept such bids directly.

Future directions for this work could include looking at tractable instances of weighted set packing to see what classes of auctions they would make tractable. It is interesting to note that while the relatively straightforward unweighted set packing problem is already NP-hard for sets of size 3, the weighted case is much more difficult than the unweighted case. Anshelevich and Karagiozova (2007) address the equivalent problem of finding matchings in a hypergraph and state that “The weighted case is significantly more complicated, however, and cannot be solved by any simple extension of the unweighted algorithm.” (weighted hypergraph matching is the weighted set packing problem, SPP).

## 7. Conclusion

We have seen what types of auctions graph matching can solve, and what types it cannot. We have also seen that those that can be handled can be solved quite quickly. In the subadditive auctions we saw that we can handle most reasonable cases of symmetric discounts over subsets of a bundle. In the asymmetric case matching was quite limited, but it must be noted that our type of symmetry is a type of redundancy. Bids must have some redundancy if they are not to be exponential in the size of the bundle on which the bid is placed. Completely incompressible bids would be exponential in the size of the bundle, and would be impractical both for the bidders to formulate and send and for the auctioneer to solve. Therefore we can expect some kind of redundancy, and our symmetric bids do seem to capture useful classes of auctions, as they respect the singleton prices and offer discounts on top of these. In conclusion we can say that matching handles subadditivity pretty well.

For the superadditive case we could really only handle pairs properly. Even seemingly straightforward “all or nothing” bids for bundles larger than two cannot be handled (and are NP-hard). Ultimately, to gain advantage from the benefits of graph matching, and to overcome its weaknesses, the graph matching approach could be combined with more general CAP solvers, such as Sandholm and Suri’s (2003).

Another role of this work could be to provide researchers in the area with valuable knowledge of the types of auction that graph matching approach are good for, and how proposed auctions might be constrained so as to be efficiently solvable by matching techniques. For example, Kothari, Parkes, and Suri (2005) describe auctions where bidders have decreasing valuations such as: the bidder offers \$10 per unit for quantity in the range  $[5, 9]$ , \$8 per unit in the range  $[10, 19]$ , \$7 in the range  $[20, 25]$  and zero for any other quantity. This is NP-hard because of the all or nothing nature of the bid for 5 units; i.e. it offers \$50 for 5 units, but nothing for 4 units. If a number of bidders make such all or nothing offers we have to solve a weighted set packing problem. There are several ways to make it tractable however; for example, make the bidder offer \$10 per unit for quantity in the range  $[1, 4]$ . This has the disadvantage that the auctioneer might sell small quantities to everyone. If it is generally

the case that tiny quantities are worthless, then it will make sense to chunk the units into fives, and make the bidder offer \$50 for the first chunk. Bidders then lose the ability to buy quantities which are indivisible by 5, however we could get around this by having two auctions, say in sequence; bulk buyers participate in the first one, and the leftover units are sold as singletons in the second. These alternatives seem to be a small compromise to achieve tractability. It is interesting to note that it is only the all or nothing nature of the first offer that makes the auction NP-hard, the bidder’s decreasing cost function poses no problem. This shows how valuable it is to have detailed knowledge of the constraints that can make auctions tractable; in many scenarios it may be practical to make a small tweak to bring enormous efficiency benefits.

We conclude with a point made by Cormen et al. (2001): algorithms are a technology. To employ graph matching algorithms to compute optimal allocations for new classes of auctions is to introduce a new technology, which can bring efficiency gains (in markets for example).

## Acknowledgements

Many thanks to Vincent Conitzer for help with NP-completeness.

## References

- Anshelevich, E., & Karagiozova, A. (2007). Terminal backup, 3d matching, and covering cubic graphs. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*. ACM.
- Balas, E., & Padberg, M. W. (1976). Set partitioning: A survey. *SIAM Review*, 18, 710–760.
- Bast, H., Mehlhorn, K., Schäfer, G., & Tamaki, H. (2004). Matching algorithms are fast in sparse random graphs. In Diekert, V., & Habib, M. (Eds.), *21st Annual Symposium on Theoretical Aspects of Computer Science (STACS-04)*, Vol. 2996 of *Lecture Notes in Computer Science*, pp. 81–92, Montpellier, France. Springer.
- Bondy, J. A., & Murty, U. S. R. (1976). *Graph Theory with Applications*. American Elsevier.
- Conitzer, V., Derryberry, J., & Sandholm, T. (2004). Combinatorial auctions with structured item graphs. In *Proceedings of the Twenty First National Conference on Artificial Intelligence (AAAI 2004)*, pp. 212–218.
- Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., & Schrijver, A. (1998). *Combinatorial optimization*. Wiley.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms* (Second edition). MIT Press.
- Dang, V., & Jennings, N. (2003). Optimal clearing algorithms for multi-unit single-item and multi-unit combinatorial auctions with demand/supply function bidding. In *ICEC ’03: Proceedings of the 5th international conference on Electronic commerce*, pp. 25–30, New York, NY, USA. ACM Press.
- de Vries, S., & Vohra, R. V. (2003). Combinatorial auctions: A survey. *INFORMS Journal on Computing*, 15(3), 284–309.

- Feder, T., & Motwani, R. (1991). Clique partitions, graph compression and speeding-up algorithms. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pp. 123–133, New York, NY, USA. ACM Press.
- Gabow, H. N., & Tarjan, R. E. (1989). Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5), 1013–1036.
- Gabow, H. N. (1983). An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pp. 448–456, New York, NY, USA. ACM Press.
- Gabow, H. N. (1985). Scaling algorithms for network problems. *J. Comput. Syst. Sci.*, 31(2), 148–168.
- Gabow, H. N., & Tarjan, R. E. (1991). Faster scaling algorithms for general graph matching problems. *J. ACM*, 38(4), 815–853.
- Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.
- Jebara, T., & Shchogolev, V. (2006). B-matching for spectral clustering. In *European Conference on Machine Learning, ECML, September 2006*, pp. 600–605. Morgan-Kaufmann Publishers.
- Kao, M.-Y., Lam, T.-W., Sung, W.-K., & Ting, H.-F. (2000). Cavity matchings, label compressions, and unrooted evolutionary trees. *SIAM J. Comput.*, 30(2), 602–624.
- Kao, M.-Y., Lam, T. W., Sung, W.-K., & Ting, H.-F. (2001). A decomposition theorem for maximum weight bipartite matchings. *SIAM Journal on Computing*, 31(1), 18–26.
- Karp, R. (1972). Reducibility among combinatorial problems. In Miller, R., & Thatcher, J. (Eds.), *Complexity of Computer Computations*, pp. 85–103. Plenum Press, New York.
- Kothari, A., Parkes, D. C., & Suri, S. (2005). Approximately-strategyproof and tractable multi-unit auctions. *Decis. Support Syst.*, 39(1), 105–121.
- Micali, S., & Vazirani, V. V. (1980). An  $O(\sqrt{|V|}|E|)$  algorithm for finding maximum matchings general graphs. In *Proceedings of the 21st Annual Symposium on the Foundations of Computer Science*, pp. 17–27. IEEE. New York.
- Müller-Hannemann, M., & Schwartz, A. (2000). Implementing weighted  $b$ -matching algorithms: insights from a computational study. *ACM Journal of Experimental Algorithms*, 5(8).
- Myerson, R. B. (1981). Optimal auction design. *Mathematics of Operations Research*, 6(1), 58–73.
- Nisan, N. (2000). Bidding and allocation in combinatorial auctions. In *ACM Conference on Electronic Commerce*, pp. 1–12.
- Penn, M., & Tennenholtz, M. (2000). Constrained multi-object auctions and  $b$ -matching. *Information Processing Letters*, 75(1-2), 29–34.
- Rothkopf, M. H., Pekec, A., & Harstad, R. M. (1998). Computationally manageable combinatorial auctions. *Management Science*, 44(8), 1131–1147. (note: 1995 tech. report at Citeseer is actually better).

- Sakurai, Y., Yokoo, M., & Kamei, K. (2000). An efficient approximate algorithm for winner determination in combinatorial auctions. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, pp. 30–37, New York, NY, USA. ACM Press.
- Sandholm, T., & Suri, S. (2003). BOB: Improved winner determination in combinatorial auctions and generalizations. *Artificial Intelligence*, 145(1-2), 33–58.
- Sandholm, T., Suri, S., Gilpin, A., & Levine, D. (2001). Cabob: A fast optimal algorithm for combinatorial auctions. In *International Joint Conference on Artificial Intelligence*, pp. 1101–1108. (IJCAI), Seattle, WA.
- Sandholm, T. (2002). Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2), 1–54.
- Sandholm, T., & Suri, S. (2001). Market clearability. In *International Joint Conference on Artificial Intelligence*, pp. 1145–1151. (IJCAI), Seattle, WA.
- Taha, H. A. (1997). *Operations research: an introduction, 6th ed.* Prentice Hall, Upper Saddle River, N.J.
- Tennenholtz, M. (2000). Some tractable combinatorial auctions. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pp. 98–103. AAAI Press / The MIT Press.
- Tennenholtz, M. (2002). Tractable combinatorial auctions and  $b$ -matching. *Artificial Intelligence*, 140(1-2), 231–243.